RICE UNIVERSITY

Live Cell Compartment Tracking: Object Tracking in Oscillating Intensity Images

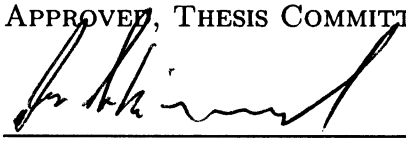
by

Kevin DeHoff

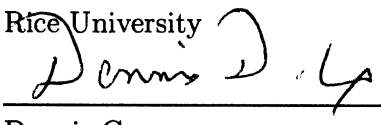
A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

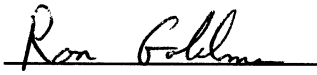
APPROVED, THESIS COMMITTEE:



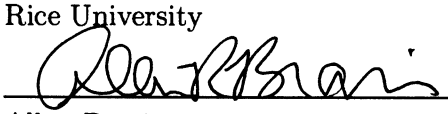
Marek Kimmel, Chair
Professor of Statistics
Rice University



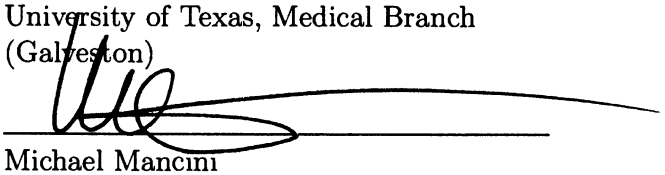
Dennis Cox
Professor of Statistics
Rice University



Ron Goldman
Professor of Computer Science
Rice University



Allan Brasier
Leon Bromberg Professor of Medicine
University of Texas, Medical Branch
(Galveston)



Michael Mancini
Associate Professor of Molecular and
Cellular Biology
Director, Integrated Microscopy Core
Baylor College of Medicine

Houston, Texas
January, 2012

ABSTRACT

Live Cell Compartment Tracking:
Object Tracking in Oscillating Intensity Images

by

Kevin DeHoff

Mathematical modeling has made great strides since the Lotka-Volterra predator-prey models. Newer models attempt to describe sub-cellular signal transduction pathways, such as the JAK-STAT and NF-κB pathways. However, the tools to accurately determine reaction and translocation rates in these pathways still have a number of drawbacks, including the effects of concentration scale on determining reaction rates and the effects of bulky additions to translocation rates. One method of overcoming these problems in signal transduction rate determination is to sample and stain cells from a full population at specific time points. However, fixed cell methods can only generate an average population rate. This could become an issue if the rate depends on the genotype of one of the proteins in the pathway.

Another method of overcoming these problems in signal transduction rates is to use unmarked nuclei in live-cell imaging techniques. However, live cell imaging methods poses different problems, primarily how to find and track nuclei and cytoplasm when cells are actively moving and the nuclear and cytoplasmic intensities are by necessity fluctuating. To date, there is only one software package designed for tracking cells under these conditions – Cell Tracker (Shen et al, 2006). Cell Tracker is designed to handle the tracking of live cell images for protein translocation studies. They

recommend using a separate color channel to mark the nucleus, although results can be obtained using unmarked nuclei. The results from Cell Tracker with unmarked nuclei are often less than optimal.

We have developed a novel segmentation scheme and variation of the particle filter algorithm to allow more accurate tracking in time series with unmarked nuclei. The proposed segmentation scheme uses a non-parametric level set algorithm to refine a fast initial thresholding step. The tracking scheme uses a dense optical flow calculation to assist the particle filter algorithm in continuing to follow the true positions of the nuclei. To test the proposed algorithm, a novel mimicry of cell movement has been developed using random perturbations of a triangular mesh structure through the use of the finite element method.

Acknowledgments

I would like to thank my primary advisor, Dr. Marek Kimmel, without whose support and guidance, this thesis would never have been started, much less completed.

I would also like to thank my committee members for taking the time out of their busy schedules to read over and evaluate my work.

In addition, I was honored to work with the faculty and staff of the Rice Statistics Department. The department members were always available for guidance and assistance throughout my tenure at Rice.

This thesis was made possible through funding by the NIH-NCI T32 grant CA096520 and in part by the Shared University Grid at Rice funded by NSF grant EIA-0216467, and a partnership between Rice University, Sun Microsystems, and Sigma Solutions, Inc.

I would like to show my gratitude to Dr. Chris Amos and the Epidemiology department at M.D. Anderson Cancer Center for their consistent support throughout my entire graduate career. In addition to the academic and statistical training I received at Rice, I was also taught to view life from the perspective of a business.

I wish to thank my parents for the love and support they have shown my entire life. I would also like to thank them for branching out of their own comfort zone to read and review this document for clarity and conciseness.

Finally, this thesis would not have been possible without the unwavering support of my wife, Lisa. She has stood by me, patiently putting her own dreams on hold while I have worked through classes, with the journey finally culminating in this project.

Kevin DeHoff

Contents

Illustrations

Tables

Chapter 1

Introduction

The goal of this project is to design an algorithm to improve the ability to track cellular compartments under particular conditions. The target result is the tracking of nuclear and cellular boundaries and the translocation of flourescent labelled proteins across these boundaries, without relying on any nuclear labelling or staining. This algorithm will allow biologists to save time in both data analysis and experimental setup when running live cell imaging experiments.

Mathematical modeling has been used in the field of biology for nearly 100 years, with the earliest known model being the Lotka-Volterra predator-prey model [43, 69]. Since then, mathematical modeling has spread into nearly every major field in the biomedical sciences [52]. One particular use is the mathematical modeling of cellular pathways. Pathways modeled include apoptosis, or programmed cell death, [10, 3, 2, 1], the cell cycle [22, 64, 20, 56], and various signal transduction pathways [74, 13, 42]. For this paper, the NF-κB signaling transduction pathway is examined.

The NF-κB pathway is often activated as a response to various cellular stimuli, including cytokines, reactive oxygen, bacterial cell walls, viral infection, and DNA damage [8]. Early work in the NF-κB pathway field was based on detecting the activation and resulting downstream effects of NF-κB in response to immunologic

stimuli, as well as how it functions in the innate immune system. However, more recent studies have shown the NF-κB pathway also plays a role in cardiovascular regulation [8] and tumorigenesis [26].

NF-κB is believed to be activated via two distinct pathways, termed the "canonical" and "non-canonical" pathways. The canonical pathway controls the activation and nuclear translocation of the primary NF-κB complex, which is made up of a heterodimer of Rel A and NF-κB1. The non-canonical pathway, on the other hand, controls the activation and translocation of the NF-κB2 complex. It should be noted that these pathways are activated by different ligands and affect different target genes [8]. The analyzed data is drawn from attempts to determine nuclear translocation rates in the canonical pathway, so future discussions will be limited to the NF-κB1 complex.

The canonical pathway controls the nuclear levels of the Rel A-NF-κB1 complex. Under normal conditions, the Rel A-NF-κB1 complex is sequestered in the cytoplasm by IκBα, preventing its nuclear migration and DNA binding activity. In the canonical pathway, IκBα is phosphorylated by an activated IKK protein, which promotes the dissociation of the NF-κB complex and lysis of IκBα. The dissociation and lysis of IκBα frees the NF-κB molecule to enter the nucleus and begin transcription of target genes. A cartoon of the described pathway is included as Figure 1.1. [8]

The basic building blocks of mathematical models of cellular pathways are systems of deterministic ordinary differential equations (ODEs) based on the mass action

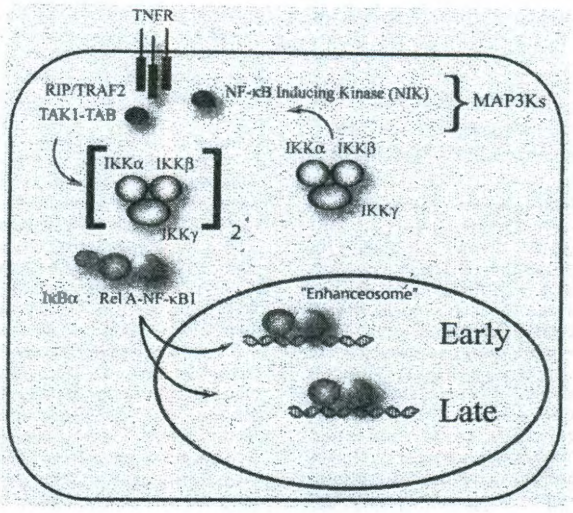


Figure 1.1 : "The canonical nuclear factor (NF)-κB activation pathway. Shown is a schematic diagram of the canonical NF-κB activation pathway downstream of the type 1 TNF receptor (TNFR1). The central regulatory point is activation of the IκB kinase (IKK), composed of the nonredundant serine-threonine kinases, IKKα and β, and the scaffolding protein IKKγ/NEMO/IKAP/FIP. Ligation of the TNFR1 induces assembly of a submembranous complex which recruits the IKK signalsome for activation. Serine phosphorylation of IKKα/β by mitogen-activated protein kinase kinase kinase (MAP3K), including the transforming growth factor (TGF)-β-activated kinase (TAK) and NF-κB-inducing kinase (NIK) results in IκB proteolysis, the Rel A-NF-κB1 heterodimer enters the nucleus to activate downstream gene network in timed temporal cascades. Depending on the stimulus duration, Rel A-NF-κB1 can undergo repeated rounds of nuclear entry and cytoplasmic redistribution in a dampened oscillatory manner." [8]

law. In these models, rapid protein-protein interactions are modeled via Michaelis-Menten formulas [42, 56, 64]. In addition to these interactions, models are also often compartmentalized into distinct nuclear and a cytoplasmic areas. Therefore, the movement of proteins between the two compartments must also be modeled, which is also done through ODEs. However, there are several drawbacks to these ODE systems. The first drawback is the difficulty in determining what functional forms are necessary to accurately model the system behavior while attempting to estimate the ODE coefficients at the same time.

The naive method of determining protein reaction rates is to use the estimated individual rates determined by *in vitro* experimentation. However, questions can be raised as to whether the rates determined in this manner are accurate for cellular systems, as there is a massive difference between the reagent concentrations required for *in vitro* experimentation and reagent concentration in cells. Similarly, there are confounders in cellular systems which do not appear in *in vitro* experimentation, such as the cytoskeleton, organelles, and additional competing proteins. As a result, biologists have begun falling back on cellular studies, especially for determining *in vivo* translocation rates.

There are two primary experimental methods to determine cellular translocation rates. The first method, used by labs such as Stephen Wong's at the Methodist Hospital Research Institute in Houston, draw a subset of the total cellular population at each time course. These cells are then fixed, stained, and then processed, which

allows for calculation of the average translocation rates. However, using the fixed cell imaging method requires assumptions about translocation rate distributions. The secondary method follows the same sample of cells for the entire time course, taking time lapse images. These images are then analyzed. The live cell imaging method allows for the exact calculation of the translocation rate for a single cell. By examining a series of cells, is is possible to calculate the distribution of translocation rates across population, which is useful for running stochastic models.

In the fixed cell microscopy studies, the nucleus and cytoplasm are often stained for easy identification, often using DAPI as a nuclear stain. However, because DAPI binds directly to cellular DNA, it has a great potential to interrupt cellular function. Therefore, DAPI staining cannot be reliably used for a time course experiment. Attaching fluorescent proteins to the cytoskeleton would likely have a similar effect, due to the excess bulk. Therefore, flourescent peptide chains are attached to the proteins of interest as a method of tracking protein location while causing the least interference with cellular processes. As an example, if the desire is to study the rate of translocation of NF-κB from the cytoplasm to the nucleus in response to external stimuli, only flourescent- labeled NF-κB proteins would be used.

Under these conditions, there are several steps for processing results. First, the locations of all the cells and their nuclear and cytoplasmic boundaries must be identified. Each cell would then have to be individually tracked to determine the rate of migration of NF-κB from the nucleus to the cytoplasm and back. It is also necessary

to determine which cells have divided, died, or migrated out of the viewing area during the course of the experiment and stop tracking those cells. The identification and tracking of the cells each pose their own problems, which will be discussed in Chapter 3.

There currently exist several free software applications for the identification and tracking of live cell images. The Cell Tracker package [61] is currently most suited for live cell image analysis, as it has a completely graphical interface, which allows the users to enhance the image to assist in manual boundary tracing. However, the manual boundary tracing algorithm only places the control points of a Bézier curve, as opposed to placing line segment points. While the estimation of boundaries using Bézier curves leads to a cleaner edge definition, the results tend to be unpredictable until a significant learning curve is mastered. In addition, although Cell Tracker does have an automatic boundary detection algorithm, the algorithm has a tendency to fail, especially when the nucleus is at the very edge of the cell, when cells are touching, and when there are no clear cellular edges. Cell Profiler [34], on the other hand, is designed for high-throughput identification and analysis of fixed cellular images, although it does have some cell tracking capability. The problem with Cell Profiler is its design as a pipeline system. A pipeline system is one which allows the user to select a series of modules and algorithms to use on all of the images in a series. However, in a pipeline system, the same parameters must be used for all of the images. Therefore, Cell Profiler in its current form would fails on live tracking data because the parameters

required to successfully process any images with dark nuclei and bright cytoplasm will fail for later images with a bright nuclei and dark cytoplasm. A third option is DCELLIQ [40], which is similar to Cell Profiler, but meant to be used primarily for cell tracking, where Cell Profiler is designed for analysis of fixed cells. DCELLIQ, however, requires bright nuclei throughout the image series, and is unable to cope with oscillating intensity regions.

This document describes an algorithm to sucessfully track live cell images without any separate nuclear markings. Our proposed algorithm is based around a two-step optical flow estimation process and a dynamic weight between edges detected in the intensity image and boundaries of regions estimated using the results of the above optical flow calculation. Additionally, the creation of a library of simulations used to test the proposed algorithm is described. A series of error metrics to be calculated for each frame from the tracking results of the simulation library is also defined. In addition, a visual comparison of results from of Cell Tracker and the proposed algorithm on the same experimental data is performed. Finally, a subjective comparison of integrated region intensities produced by the proposed algorithm to those generated by a completely manual tracking is performed.

Chapter 2

Background

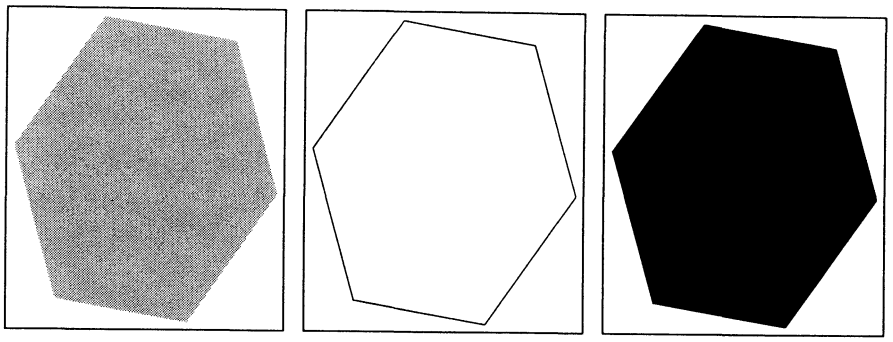
The field of computer vision is a discipline based on the construction of meaningful descriptions of physical objects from images. For example, computer vision is the attempt to feed in an image made up of several horizontal, vertical, and diagonal lines, and have the computer report that the subject of the image is a house. The steps in computer vision include identifying primitive features in the image, such as lines, corners, and regions, and then associating the layout of these primitive features with a known library of objects. This process is the basis of image segmentation, which will be discussed in Chapter 2.1.

There are many applications in which merely recognizing an object in a static frame is insufficient. Some examples of these applications include automatic security monitoring, face recognition, and live cell imaging, as we have discussed before. In these applications, the computer needs to not only identify the object in the initial image, but also to be able to find the same object in a subsequent image. An overview of object tracking algorithms will be given in Chapter 2.2.

2.1 Image Segmentation

Image segmentation is the process of partitioning an image into a number of discrete regions that fulfill two primary requirements. First, each pixel in the image must be used in exactly one region. Second, every region must be a fully connected set. In monochrome (greyscale) images, there are two primary methods of segmenting an image. [24]

The first segmentation method is by using edges - local discontinuities in image intensities. These methods are based on finding the local maxima of intensity gradients in each direction. Edge detectors are algorithms that result in a binary image, with the foreground being the detected edges in the image. As an example, the classic edge detection algorithm is the Canny edge detector. The Canny edge detector begins by smoothing the image intensities. The intensity gradient magnitude and direction are calculated at each pixel in the smoothed image. Non-maximal gradient values in the calculated direction are suppressed. A final edge map is generated by a hysteresis thresholding step, where any gradient magnitude greater than some value T_1 is set as foreground, and any gradient magnitude greater than some lower value T_2 and touching a previously set foreground pixel is also set to foreground [11]. It is assumed that the detected edges are region boundaries and the pixels to either side are therefore assigned as separate regions. For example, Figure 2.1 shows how edge maps can be used to determine region boundaries when the region intensities are fairly smooth. Edge segmentation methods are, however, limited to instances where

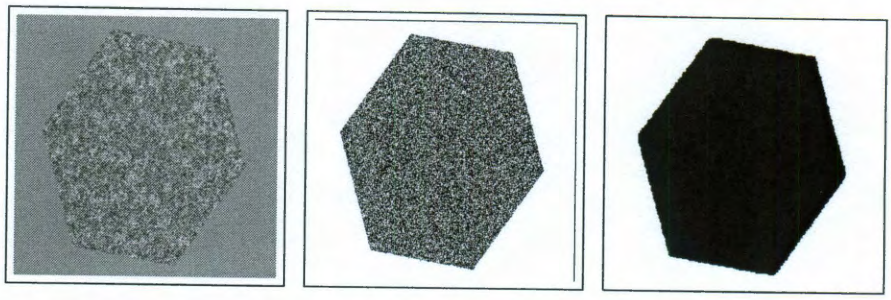


(a) Original Image (b) Edge Image (c) Defined Regions

Figure 2.1 : Defining regions using edge images

there is a clear distinction between region intensities.

The second segmentation method is through defining a region descriptor and separating the image into component regions based on the descriptor value. As an example, examine Figure 2.2. In Figure 2.2a, a highly textured region was placed against a solid field. In highly textured images, such as the one as shown here, an edge detector would not only find edges at the region boundary, but also place edges throughout the interior of the textured region, as seen in Figure 2.2b. It would therefore be impossible to use edges to segment the region. The two regions could still be segmented if a pixel is defined to be in the "foreground" region if the variance of the intensities in a local neighborhood was sufficiently high, as shown in Figure 2.2c.



(a) Original Image (b) Edge Image (c) Defined Regions, using $\sigma > 10$ under a 7×7 window

Figure 2.2 : Segmenting textured images

2.2 Object Tracking

There are two main classes of algorithms involved in tracking objects through time series images. These classes can be considered forward-looking algorithms and backward-looking algorithms. Forward-looking algorithms tend to include such algorithms as the particle filter [30, 19], the mean-shift algorithm [16], and algorithms based on active contours [38, 17, 68]. All of these algorithms algorithmically estimate the new position from either an estimate of or the exact previous position. That is, the position of object O at time $t+1$ can be determined from the position of the same object O at time t.

Backward-looking algorithms, on the other hand, have as a general theory to segment and detect objects at both time t and $t+1$. After both objects have been

identified, they are associated based on some statistic, often centroid distance, pixel intensity distribution, or approximate shape [32, 34].

2.3 Previous Work

There are currently three free cell tracking programs available for use as well as several proposed algorithms. The software packages are Cell Tracker, by the Bioanalytical Sciences group at the University of Manchester [61], Cell Profiler, developed by the Broad Institute [34], and DCELLIQ produced at the Methodist Research Hospital [40]. Each of these packages is a well-established application, but each also have their drawbacks.

2.3.1 Cell Profiler

Cell Profiler is designed as a high-throughput static image analysis program. In Cell Profiler, a pipeline of image processing methods is assembled and automatically run on a large number of images using the same parameters for every image. Cell Profiler works under several assumptions. Cell Profile first assumes background lighting and cell size to be similar for every image examined. Cell Profiler also assumes the relative intensities for each region to be constant, i.e. the nucleus is always either lighter or darker than the surrounding cytoplasm. This assumption arises because the tracking pipeline involves the initial identification of each nucleus as a "primary object" and finding the surrounding cytoplasm as a "secondary object." For fixed cell images,

the two-step segmentation as described here is a very fast and efficient method of identifying individual cells. However, as it is unknown for each time period whether the nucleus is brighter or darker than the surrounding cytoplasm, Cell Profiler quickly becomes inadequate for analyzing our data.

Cell Profiler also has several simplistic tracking algorithms included in the package. These tracking algorithms are backward-looking algorithms, which identify objects in every frame and correlates them. Cell Profiler allows tracking correlation by object overlap, object position, and object measurement. Object overlap is simply the amount of spatial overlap between consecutive frames. Object position links objects by a "nearest neighbor"-type algorithm. Object measurement links objects based on some predefined measurement. Examples in the program documentation include object intensity and object shape. Cell Profiler also includes a more advanced linear assignment problem framework based on tracking sub-cellular particles [32].

These algorithms are unsuitable for a variety of reasons. First, the object overlap and distance algorithms are potentially unsuitable due to the speed of the moving cells in each experiment. In fact, because time-lapse data is being examined, there is no reason to assume the cellular area overlaps between individual frames. The object position linking is potentially unsuitable for a similar reason. If there is a high concentration of cells in the image, there is no guarantee this simple algorithm will actually link a cell in the first image with the same cell in a subsequent image. The object measurement algorithm has some possibility of being useful in our tracking

situation. For most data, it would be impossible to track by shape, as plated cells are amorphous, that is they have no guaranteed shape. Nuclei could be tracked by shape, as they are known to be roughly elliptical, but it is also quite possible for the nuclei to be easily confused for each other. Tracking by intensity measurements would be impossible, as the goal of the experiment is the movement of tagged particles in and out of the nucleus, indicating the nuclear and cytoplasmic intensities are in a constant state of flux. The linear assignment algorithm is likewise unsuitable, because it relies on the particles being a reasonably constant intensity over time.

2.3.2 Cell Tracker

In contrast to Cell Profiler's attempt to be a semi-automated system, Cell Tracker is designed to be a graphically interactive program. Cell Tracker includes multiple image processing features with the intention of enhancing the image boundaries and simplifying the selection of nuclear and cytoplasmic boundaries. Cell Tracker uses several steps to identify the important boundaries in the image in its automatic algorithms. First, the texture of the cellular image is smoothed to set all of the background to a similar intensity and all of the foreground to another intensity. The smoothing allows for the selection of a global threshold value for the image. Once the image is thresholded, the cytoplasm and the nuclei of the various cells are identified.

There are several problems with the automated procedure. First, it is quite possible for a nucleus to be touching the outside of a cell. In these instances, the cellular

and nuclear boundaries are not correctly determined. In many cases, there is no detectable edge between the nucleus and the background, causing the failure of the algorithm. Another problem with the Cell Tracker identification algorithm is the failure to incorporate any preprocessing algorithms applied to the image, starting from the initial image as loaded.

Cell Tracker also includes the ability to have users manually trace cellular and nuclear boundaries. While an experienced user employing the manual tracking method will always yield the best results, the interface is difficult to manage. Cell Tracker uses cubic B-splines to mark and track the cellular and nuclear boundaries. However, the program interface for drawing these splines involves placing only control points. The resulting boundary can often be rather different from what a user expects.

Cell Tracker's tracking algorithms are fairly complete and robust. There are two tracking algorithms implemented in this software. The primary algorithm is the particle filter algorithm, a forward-looking algorithm which scores a number of candidate objects according to "features" in the subsequent image. Each of these objects is assigned a likelihood based on the calculated score, and the expected value is taken as the "true" position in the next frame. However, the current implementation of the particle filter algorithm is dependent on how well the "features" (often edges) of the next frame match with the candidate objects. Due to oscillating intensity conditions, there are times where nuclear edges simply do not exist, due to a near homogeneous intensity throughout the entire cell. In these situations, the particle filter algorithm

fails, and the position of the nucleus is lost for several frames. In Cell Tracker, the "basic" cell tracking algorithm only tracks the nuclei, and re-estimates the cytoplasm boundaries in each frame based on a Voronoi diagram [33].

The second tracking algorithm in the Cell Tracker software is based on the identification of key frames in the time series. In the key frame tracking algorithm, at least two frames must be fully processed, meaning all necessary cellular and nuclear boundaries must be identified and labeled in both frames. The key frame tracking algorithm assumes the shape of the boundary can be linearly interpolated between these key frames.

Due to other factors, such as cell division and death, users must also page through the experimental results one image at a time, removing cells which have died or divided, as well as correcting tracking errors. Due to the errors in tracking steps where the nuclear and cytoplasmic intensities are roughly equal, as well as the post-processing step, it is not uncommon for the entire processing time of a 24-hour experiment to be nearly the same amount of time (in man-hours).

2.3.3 DCELLIQ

The DCELLIQ software package seems to be closer to Cell Profiler than Cell Tracker. In DCELLIQ, cells are identified first by their nuclei, which are classified into dark nuclei, light nuclei, and background by a multiple adaptive thresholding technique after background correction [41]. The nuclei are detected using Gaussian filtering

and a seeded watershed algorithm [41]. Once the watershed is complete, they score each potential nucleus based on nine easily calculated features, including the area, eccentricity, and axis length. These scores are based on a training set of well-segmented nuclei. Nuclei which have score less than some threshold, are assumed to be an under-segmented collection of nuclei. DCELLIQ splits these nuclei by taking the two points located in the $\frac{1}{4}$ and $\frac{3}{4}$ of the major axis. DCELLIQ repeats the seeded watershed algorithm using these points as centers. These new nuclei are accepted if the scores of the newly generated nuclei are greater than the score of the split nucleus. DCELLIQ continues to split nuclei with score less than the given threshold until no new nuclei are generated [41].

The tracking algorithm employed in DCELLIQ is a backwards-looking algorithm, where the overall goal is the association of nuclei in frame t with their previous position in frame $t-1$. The cells for each frame are simultaneously associated using integer programming optimization [40]. The association method is essential to their algorithm due to the high number of cells in the image, as well as to the similarity in shape of nuclei.

2.3.4 Other Works

Shen et al's segmentation and tracking algorithm [60] is similar to both Cell Profiler and Cell Tracker. Shen et. al uses K-means clustering on the cellular intensity for the initial segmentation, assuming the nuclei to always brighter than the cytoplasm,

which is always brighter than the background [60]. Because Shen et. al are not working with oscillating intensity images, the K-means clustering provides a fast and accurate segmentation method.

The tracking method employed by Shen et. al is a forward-looking algorithm which takes into account the shape of the cytoplasm to determine in which direction the cell will be moving. Shen et. al leverage the knowledge that moving fibroblast cells maintain a roughly triangular shape during movement [60]. To determine the direction of movement, an ellipse is fit to the cytoplasmic boundary - the major axis reveals the line upon which the cell is moving. The ellipse is rotated until the major axis is vertical. Two isosceles triangles are fit to the ellipse, one which points upward with its base being the bottom of the rectangular bounding box of the cell, and the second pointing downward with its base being the top of the bounding box. The area of overlap between the cell and each of these two triangles are calculated, with these overlaps being defined as the "Up index" for the upward-facing triangle and the "Down index" for the downward-facing triangle [60]. If the smaller index is calculated to be more than 70% of the larger index, the cell is assumed to be stationary. Otherwise, the direction of movement is assumed to be the direction of the larger index.

The most prevalent software packages which can be used for live cell tracking have been described. In addition, it has been shown why Cell Profiler is unsuitable for tracking cells with unstained nuclei, which is a requirement for translocation rate

calculation. The DCELLIQ program and the algorithm by Shen et. al both require consistently stained nuclei to sufficiently track the cells, and as a result are unsuitable for examining translocation data. Cell Tracker does a far better job of identifying and tracking cells with unstained nuclei. However, there are many conditions under which the identification and tracking results are far from optimal, due to the sole use of edges. Due to the potential low contrast of the images and the occasional lack of a visible nuclear boundary, a strict use of edges will nearly always produce sub-optimal results, even from the best hand-drawn nuclear and cellular boundaries.

Chapter 3

Cell Tracking

The proposed algorithm is a multi-step process to detect and track nuclear and cytoplasmic boundaries during protein translocation experiments in the absence of any nuclear marking. The experiment is assumed to be synchronized in such a way so the nuclear and cytoplasmic boundaries are distinguishable in the segmentation frame. The algorithm can track the nuclear and cytoplasmic boundary locations either forward or backward through the remaining images in the time series.

The algorithm as a whole is summarized in Figure 3.1, a flowchart which shows the different steps taken, the permanent and temporary data used and produced, and where each piece of data is used in the identification and tracking algorithm. Figure 3.1 shows the first step as an initial cellular detection on Frame 0. For each subsequent frame (considered the current frame), it is assumed that the frame immediately prior (considered the previous frame) is correctly tracked. The average change in intensity for pixels in each region between the previous and current frame is estimated. For this and future estimations, the regions are considered to be background, unidentified foreground items, and each cellular compartment The average region intensity change is used to calculate the optical flow for each pixel based on both the final region assignments in the previous frame and the spatial and temporal gradients between

the previous and the current frame. Using the calculated dense optical flow field and the 95% confidence intervals in each direction, region assignments for each pixel in the current frame are estimated. In addition, the edges from the raw data for the current frame are calculated using the Canny edge detector [11]. These calculated edges, along with the boundaries from the estimated regions, are used to estimate the five-parameter ellipses which will represent the new location of each nuclear region being tracked. These parameterized ellipses are used with the raw intensity data from the current frame to estimate the cytoplasmic boundaries for each cell.

Chapter 3.1 discusses the theory behind the calculation of the optical flow vector field as well as several of the seminal works in the field pertaining to the proposed algorithm. In particular, the Horn-Schunck optical flow calculation algorithm [29] is discussed, which is one of the earliest optical flow calculation algorithms. A second primary optical flow calculation is the Lucas-Kanade algorithm [44], which will be modified slightly for later use. Finally, a physics-based optical flow algorithm by Haussecker and Fleet [28] will be explained. Haussekcer and Fleet's physics-based model will also be used in the proposed algorithm, as it allows for a structured relaxation of a key constraint in optical flow calculations. Chapter 3.2 describes the theory behind the particle filter algorithm that will form the basis for final estimation of the nuclear parameters.

Chapter 3.3 describes the methods used for the initial segmentation (the determination of region assignments in the first image). Chapter 3.4 describes the actual

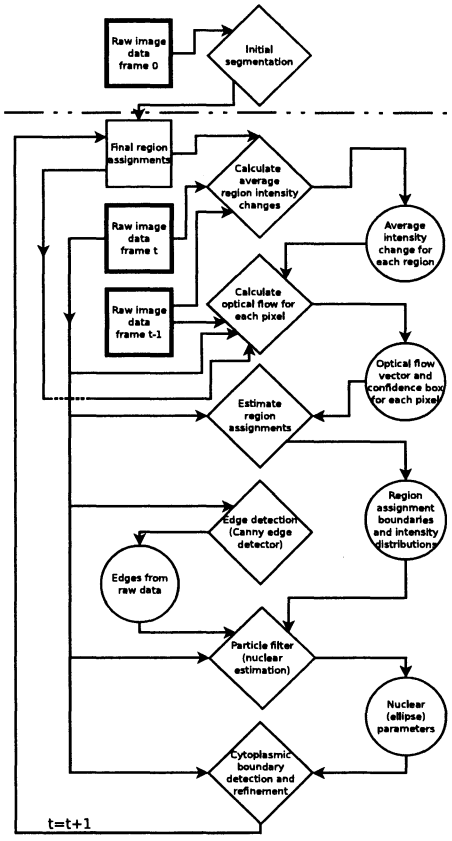


Figure 3.1 : A flowchart for the proposed tracking algorithm. Diamond entries are calculation and estimation steps. Square entries are permanent data - i.e. data that are provided by the user or data outputted by the program. Round entries are temporary data - i.e. data that are only relevent for calculation of a subsequent step in the current frame. Boldface squares are user-provided image data.

tracking algorithm in far more detail, explaining and justifying each step taken. Finally, Chapter 3.5 outlines the different error metrics to be used when comparing the proposed algorithm and Cell Tracker's results against a library of simulations, the generation of which is given in Chapters 4 and 5.1.

3.1 Optical Flow Algorithms

Optical flow is defined as the apparent velocities of moving objects inside an image boundary [29]. Generally, the calculation of optical flow is based on the gradient of the image intensity between frames of an image sequence. However, this is an ill-posed problem, as the intensity gradient is a single dimension, but the optical flow itself is a two-dimensional vector field. Therefore, several constraints must be set to calculate a unique optical flow representation. One of the primary constraints is the brightness constraint, given as

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t).$$

Examination of the first-order Taylor series of the brightness constraint leads directly to the brightness constraint equation (3.1).

$$
\begin{aligned}
I(x,y,t) &= I(x+\delta x, y+\delta y, t+\delta t) \\
&= I(x,y,t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t \\
0 &= \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t \\
&= \frac{\partial I}{\partial x}\frac{\delta x}{\delta t} + \frac{\partial I}{\partial y}\frac{\delta y}{\delta t} + \frac{\partial I}{\partial t} \\
&= \frac{\partial}{\partial x}I(x,y)\boldsymbol{v}_x(x,y) + \frac{\partial}{\partial y}I\boldsymbol{v}_y(x,y) + \frac{\partial}{\partial t}I(x,y), \qquad (3.1)
\end{aligned}
$$

where $I(x,y)$ is the pixel intensity at the image location (x,y), and $\boldsymbol{v}(x,y)$ is the optical flow vector at the location (x,y) [29, 21, 28]. However, even with the brightness constraint, the calculation of the optical flow field for a single pixel is still an ill-posed problem [21]. There are several basic constraints used to calculate optical flow, all of which assume a local smoothness around the pixel in question.

The algorithm by Horn and Schunck [29] assumes smoothness in the Laplacian of the spatial gradient and optical flow field around the pixel being examined, while the Lucas-Kanade [44] and Haussecker and Fleet [28] assume the optical flow vectors in a neighborhood of the pixel in question are constant.

3.1.1 Horn And Schunck

Horn and Schunck's algorithm, as previously stated, is based on both the brightness constraint equation (3.1) and the assumption of smoothness in the Laplacian of both the spatial gradient and optical flow field around a given pixel. The Horn-Schunck

algorithm uses forward differences to estimate the image gradient with respect to x, y, and t. Horn and Schunck estimate the errors

$$\xi_b = I_x \boldsymbol{v}_x + I_y \boldsymbol{v}_y + I_t, \text{ and} \tag{3.2}$$

$$\xi_c = (\bar{\boldsymbol{v}}_x - \boldsymbol{v}_x)^2 + (\bar{\boldsymbol{v}}_y - \boldsymbol{v})^2, \tag{3.3}$$

where I_i is the forward difference of the image intensity in the i direction, ξ_b measures the error from (3.1) and ξ_c measures the departure from smoothness in the optical flow. The total error is given by

$$\xi = \alpha^2 \xi_c^2 + \xi_b^2, \tag{3.4}$$

where α^2 is a weighting factor. By differentiation of (3.4), Horn and Schunck devised an iterative scheme to calculate $\boldsymbol{v}$, given by

$$\boldsymbol{v}_x^{n+1} = \bar{\boldsymbol{v}}_x^n - I_x \frac{I_x \bar{\boldsymbol{v}}_x^n + I_y \bar{\boldsymbol{v}}_y^n + I_t}{\alpha^2 + I_x^2 + I_y^2}, \text{ and} \tag{3.5}$$

$$\boldsymbol{v}_y^{n+1} = \bar{\boldsymbol{v}}_y^n - I_y \frac{I_x \bar{\boldsymbol{v}}_x^n + I_y \bar{\boldsymbol{v}}_y^n + I_t}{\alpha^2 + I_x^2 + I_y^2}. \tag{3.6}$$

[29]

A major advantage of the Horn-Schunck method is that due to its iterative nature it can calculate a fully dense optical flow field for the entire image, even in the presence of large uniform regions. This is in contrast to the Lucas-Kanade and Haussecker and fleet algorithms, which fail in regions of uniform intensity, as will be discussed in Chapters 3.1.2 and 3.1.3.

3.1.2 Lucas-Kanade

Lucas and Kanade's algorithm was originally based on performing image registration [44]. In image registration, two images are assumed to be identical with the exception of some spatial morphing function. The goal of image registration is to determine the morphing function which will yield the second image from the first [75]. Image registration is very similar to the calculation of optical flow, with the exception that optical flow holds over a small region, while image registration is generally used for the entire image. In Lucas and Kanade's method, the flow vectors are assumed to be constant in the neighborhood of a pixel [21]. Under this assumption, let $\boldsymbol{q}$ be a vector of pixels which make up the neighborhood of p, the pixel in question. It is possible to derive a linear system of equations from (3.1), such as

$$I_x(q_1)\boldsymbol{v}_x + I_y(q_1)\boldsymbol{v}_y = -I_t(q_1)$$

$$I_x(q_2)\boldsymbol{v}_x + I_y(q_2)\boldsymbol{v}_y = -I_t(q_2),$$

and so forth. In matrix form, however, this simplifies to

$$\begin{pmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_x \\ \boldsymbol{v}_y \end{pmatrix} = \begin{pmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \end{pmatrix}. \tag{3.7}$$

Now, if the variations from the brightness constraint are assumed to be independent and identically distributed Normal variates, a least-squares framework can be used to estimate $\boldsymbol{v}$ [21].

The Lucas-Kanade algorithm, being a strictly local method, however, fails to accurately calculate the optical flow under several circumstances. The first is regions of uniform intensity. In these instances, the system becomes underdetermined, and $\boldsymbol{v}$ cannot be recovered. Additionally, in the presence of severe violations of the brightness constraint, the results obtained from the Lucas-Kande method are highly suspect [21].

3.1.3 Haussecker and Fleet

Haussecker and Fleet's algorithm relaxes the brightness constraint (3.1), which can also be written as

$$I(\boldsymbol{x}(t), t) = c, \tag{3.8}$$

where $\boldsymbol{x}(t)$ is a pixel path and $I(\boldsymbol{x}(t), t)$ is the intensity value of the path at time t. Now, if we allow the right hand side of (3.8) to be a function of t, it is possible to determine not only the optical flow vector but also the rate of change of intensity [28]. By taking this approach, let

$$I(\boldsymbol{x}(t), t) = h(g_0, t, \boldsymbol{a}),$$

where g_0 is the intensity in the initial image, and $\boldsymbol{a}$ is some set of parameters which define the brightness change model. It is possible to modify (3.1) to be

$$\frac{\partial}{\partial x} I(x, y)\boldsymbol{v}_x(x, y) + \frac{\partial}{\partial y} I\boldsymbol{v}_y(x, y) + \frac{\partial}{\partial t} I(x, y) = \frac{\partial}{\partial t} h(g_0, t, \boldsymbol{a}). \tag{3.9}$$

However, Haussecker and Fleet also notice it is possible to write any function h as a Taylor series out to Q terms, meaning that h is always analytic in $\boldsymbol{a}$ [28]. From (3.9),

they are able to write

$$\frac{\partial}{\partial t}h(g_0, t, \boldsymbol{a}) = f(g_0, t, \boldsymbol{a}) = (\nabla_{\boldsymbol{a}} f)^T \boldsymbol{a}. \tag{3.10}$$

If $\boldsymbol{p}$ is defined as

$$\boldsymbol{p} = [-\boldsymbol{a}^T, \boldsymbol{v}^T, \mathbf{1}]^T \tag{3.11}$$

and

$$\boldsymbol{d} = [(\nabla_{\boldsymbol{a}} f)^T, \nabla g^T, g_t]^T, \tag{3.12}$$

then by (3.9),

$$\boldsymbol{d}^T \boldsymbol{p} = 0. \tag{3.13}$$

Haussecker and Fleet follow Lucas-Kanade's example and assume $\boldsymbol{p}$ is constant over a local region in both space and time. By this assumption, it is possible to form a neighborhood of size N around the pixel in question, such that the total constraint becomes

$$\boldsymbol{G}\boldsymbol{p} = \mathbf{0},$$

where

$$\boldsymbol{G} = [\boldsymbol{d_1}, \ldots, \boldsymbol{d_N}]^T. \tag{3.14}$$

If the error noise in the measurements of I are assumed to be independent and identically distributed Normal variates, the maximum likelihood estimator of $\boldsymbol{p}$ is the total least squares estimator, with estimate

$$\frac{\boldsymbol{p}^T \boldsymbol{G}^T \boldsymbol{p}}{\boldsymbol{p}^T \boldsymbol{p}}. \tag{3.15}$$

[28]

The advantage of the Haussecker and Fleet method is closer optical flow calculation for rotating objects and objects in which the image is undergoing a structured intensity change. Examples of structured intensity changes would be an object undergoing diffusion or an object rotating through different brightness areas [28]. However, Haussecker and Fleet have stated that in order for this algorithm to work correctly, the form of the underlying brightness change model must be known [28], and they have made no attempts to attempt a model search to determine the best brightness flow model for any given situations.

3.2 Particle Filter

The particle filter algorithm was originally developed in 1993 by Gordon et. al as a means of estimating posterior distributions under certain difficult conditions [25]. The first known use of the particle filter as a tracking algorithm was the Condensation algorithm [30]. The Condensation algorithm uses the particle filter simulation method to determine the posterior distribution of contours given the edges in the current frame.

In the base particle filter tracking algorithm, the previous location is updated via independent samples from some underlying movement distribution, which is often constant. A new potential position is calculated as a random variate in some other parameter (direction, rotation angle, etc). The weighted average of a number of these

potential positions is used as the estimate of the new position.

The primary advantage to the particle filter algorithm is the lack of requirement for normality in any of the explicit or implicit distributions. As a result, the particle filter has a much greater degree of flexibility than the Kalman Filter [30], one of the other leading stochastic tracking algorithms.

3.3 Cell Segmentation

The first step in any object tracking algorithm is the determination of the positions of the objects to be tracked in the initial frame. The initial cellular detection is described in Chapters 3.3.2-3.3.4. Once the cells have been detected, the proposed algorithm will use a modified particle filter algorithm [30] to track them between the different frames. This algorithm is described in detail in Chapter 3. Figure 3.2 shows an example frame in which the cells will be detected.

The image in Figure 3.2, appears very noisy. As a result, a median filter is used to smooth out the speckled appearance. Smoothing the image will assist in edge detection. The smoothed image is segmented into foreground and background based on a local thresholding algorithm. The details of the smoothing and local thresholding are given in Chapter 3.3.1.

It is possible for images with high background noise to develop a large number of extraneous foreground pixels throughout areas which should be classified as background. To alleviate this issue, the proposed algorithm refines the boundaries of

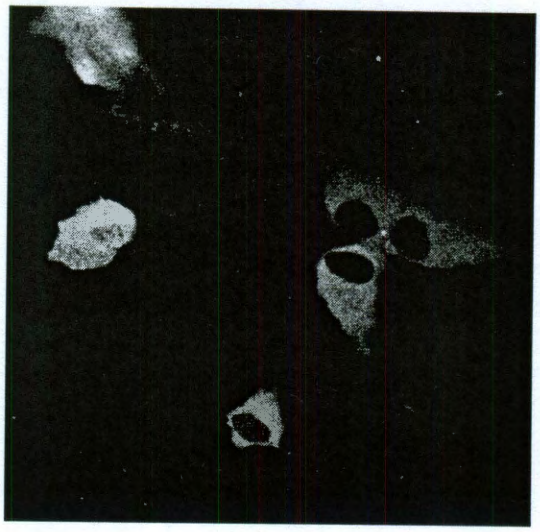


Figure 3.2 : Base image for examples in Chapter 3.3

what is considered foreground and background using a level set algorithm, which is described in Chapter 3.3.2. The result from the level set algorithm is used as a mask, removing foreground pixels which are too far away from what is believed to be cytoplasmic boundaries.

Edges are detected in the resulting binary image using basic using basic morphological algorithms described in Chapter 3.3.1. These boundaries are considered to be an edge image. The edge image is used to detect ellipses in the image, which are assumed to correspond to nuclear boundaries. The details of the ellipse detection are given in Chapter 3.3.3. Once the nuclei have been detected, the level set mask is used to determine the boundaries of the cytoplasmic region as described in Chapter 3.3.4.

3.3.1 Generic Algorithms

The basic algorithms used in image processing are set-based operations. These include thresholding, erosion, dilation, and various blurring operations. More advanced algorithms such as edge detection use a combination of these different basic operations.

There are three primary blurring operations that are often used in image processing. Each of these operations maps an integer greyscale value in image $\boldsymbol{I}$ to another value in $\boldsymbol{J}$ given by the relation

$$\boldsymbol{J}_{i,j} = g\big(\mathcal{N}(\boldsymbol{I}_{i,j})\big),$$

where $\mathcal{N}$ denotes a neighborhood of points around $\boldsymbol{I}_{i,j}$. The most common blurring operation is simple linear averaging, where

$$g\big(\mathcal{N}(\boldsymbol{I}_{i,j})\big) = \frac{\sum \mathcal{N}(\boldsymbol{I}_{i,j})}{\big|\mathcal{N}(\boldsymbol{I}_{i,j})\big|}.$$

Another common blurring operation is Gaussian smoothing, where the points are weighted by the density of an appropriate bivariate normal distribution centered at the point (i, j). The third primary blurring operation is the median filter. In this operation,

$$\boldsymbol{J}_{i,j} = \operatorname{med}\big(\mathcal{N}(\boldsymbol{I}_{i,j})\big).$$

Each of these different operations yield a slightly different result. The median filter is used in the proposed algorithm, as the application of the filter does not affect the location of the edges. It is known that the linear averaging and the Gaussian

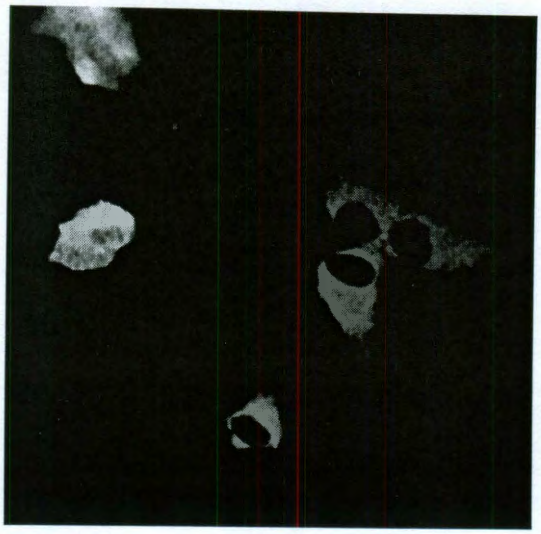


Figure 3.3 : Results of 5×5 median filter on Figure 3.2

filter both tend to shift the edge locations, yielding inappropriate results. Although several filter shapes, such as circular, cross, and X-shaped filters are useful in many situations, a square neighborhood is chosen for algorithmic simplicity. Figure 3.3 shows the result of a 5×5 median filter on the base image shown in Figure 3.2.

Thresholding operations are fairly basic operations used to map an integer greyscale value to a binary value. Global thresholds assume a single value T for the entire image, where the map $\boldsymbol{I} \mapsto \boldsymbol{J}$ for a given pixel (i, j) takes on the form

$$\boldsymbol{J}_{i,j} = \begin{cases} 1, & \boldsymbol{I}_{i,j} > T; \\ 0, & \boldsymbol{I}_{i,j} \leq T. \end{cases}$$

Figure 3.4 shows an example of using a global threshold value T of 36 on Figure 3.3. As can be seen, the low contrast extremities of the cytoplasm are incorrectly classified

Figure 3.4 : Results of global thresholding Figure 3.3 with $T = 36$

as background.

A second possibility for thresholding is to use an adaptive, or local threshold. For a local thresholding operation, T is different for each pixel. In the majority of implementations, T is calculated by

$$\boldsymbol{T}_{i,j} = \frac{\sum \mathcal{N}(\boldsymbol{I}_{i,j})}{|\mathcal{N}(\boldsymbol{I}_{i,j})|}.$$

Figure 3.5 shows the result of a local threshold using a 9×9 square neighborhood to calculate $T_{i,j}$. As can be seen, the local threshold performs better in determining the true edges of the cytoplasm, but it fails to segment the inner areas of the cytoplasm due to the varied intensity.

Figure 3.5 : Results of local thresholding Figure 3.3 with a 9×9 neighborhood

3.3.2 Level Sets

Level sets were originally designed as a tool to solve Hamilton-Jacobi equations for situations where a front propagated with a speed dependent on its own curvature [57]. Some classical examples of curvature-dependent propagation are crystal growth, flame fronts and fluid boundaries [57]. Level set problems are solved by embedding the N-dimensional solution in an $(N+1)$-dimensional surface. This allows for easily varying topological features, such as breaking and merging. In level-set solutions, the N-dimensional front C is represented as the zero level set of the $(N+1)$-dimensional function ϕ, i.e.

$$C(t) = \{(x, y) | \phi(x, y, t) = 0\}. \tag{3.16}$$

In practice, $\phi(x, y, t)$ is implemented as a signed distance function from an initial

level set $\mathcal{C}$, where $\phi(x, y, t)$ is generally set as the absolute Euclidean distance from the nearest point on $\mathcal{C}$, and the sign of $\phi(x, y, t)$ is dependent on whether it is inside or outside of the front. At each time iteration, $\phi(\cdot)$ is updated based on the level set equation

$$\frac{\partial \phi}{\partial t} + F|\nabla \phi| = 0, \tag{3.17}$$

where F is a speed function dependent on the image information. [39]

One of the primary drawbacks of the standard formulation is a tendency for ϕ to develop topologies which will make further computation highly inaccurate ([39, 57]). In these situations, ϕ has to be re-initialized, which is a computationally expensive procedure which has the potential side effect of shifting the current position of $\mathcal{C}$. In fact, it is possible for the re-initialization of ϕ to completely fail [39]. To avoid dealing with these issues, the proposed algorithm uses a modification of a level set formulation first proposed by Li et al [39]. In this formulation, the speed function F is decomposed into an internal and an external gradient flow

$$F = \mu \frac{\partial \mathcal{E}_{\text{int}}}{\partial \phi} + \nu \frac{\partial \mathcal{E}_{\text{ext}}}{\partial \phi}. \tag{3.18}$$

According to Li et al., not only will a signed distance function have the property $|\nabla \phi| = 1$, but any function ϕ which satisfies $|\nabla \phi| = 1$ is the signed distance function plus a constant. Therefore, an internal energy is defined to be a penalty on how different ϕ is from a signed distance function. The described penalty is given by

$$\mathcal{E}_{\text{int}} = \mathcal{P}(\phi) = \int_{\Omega} \frac{1}{2}(|\nabla \phi| - 1)^2 dx dy, \tag{3.19}$$

where Ω is the entire image. This internal energy will automatically maintain ϕ close to a signed distance function, eliminating the need for the costly reinitialization procedure [39]. It can be further shown that

$$\frac{\partial \mathcal{E}_{\text{int}}}{\partial \phi} = -\left[\Delta\phi - \text{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right)\right]. \tag{3.20}$$

Because many cellular images can be low-contrast, a two-step process is used to segment the images. The first step is the use of a local threshold to determine initial classifications of foreground and background pixels in the image. One advantage to the local threshold algorithm is its results in highlighting all of the small changes in intensity levels. This will allow the proposed algorithm to pick out the ellipses corresponding to the nuclei fairly easily. However, if there is a high amount of background noise, it is likely that most of the background will also be seen as foreground. The local threshold will be cleaned with using the results of the level set algorithm as a mask to assist in determining the actual location of foreground objects.

To compensate for the low constrast present in many cellular images, a nonparametric level set formulation proposed by Kim et al. [35] is used. For this formulation, the image intensity for pixel x is assumed to be a random variable $G(x)$ drawn from the density p_1 if x is in Region 1 (R_1) or from p_2 if x is in Region 2 (R_2), where p_1, p_2, R_1, and R_2 are unknown. Now, the level set solution is an attempt to find a boundary $\mathcal{C}$ to correctly separate the image into R_1 and R_2. In this case, a potential curve C separates the image domain Ω into a binary label $L_C = \{L_1, L_2\}$ using the

mapping

$$L_C(x) = \begin{cases} L_1, & x \in \hat{R}_1; \\ L_2, & x \in \hat{R}_2. \end{cases}$$

Kim et al. [35] showed the mutual information between $G(x)$ and C to be represented by

$$\begin{aligned} I(G(x); L_C(x)) =& h(G(x)) - P(L_C(x) = L_1)h(G(x)|L_C(x) = L_1) \\ & - P(L_C(x) = L_2)h(G(x)|L_C(x) = L_2), \end{aligned} \tag{3.21}$$

where $h(Z)$ is the differential entropy of a continuous random variable Z. They also showed that (3.21) is maximized only if $C = \mathcal{C}$. Because R_1 and R_2 are unknown, $I(G(x); L_C(x))$ must be estimated. The gradient flow for updating C can be shown to be given by

$$\begin{aligned} \frac{\partial C}{\partial t} =& \left[\log \frac{\hat{p}_1(G(C))}{\hat{p}_2(G(C))} + \frac{1}{|\hat{R}_1|}\int_{\hat{R}_1} \frac{K(G(x) - G(C))}{\hat{p}_1 G(x)} dx \right. \\ & \left. - \frac{1}{|\hat{R}_2|}\int_{\hat{R}_2} \frac{K(G(x) - G(C))}{\hat{p}_2 G(x)} dx \right] \vec{N} - \alpha\kappa\vec{N}, \end{aligned} \tag{3.22}$$

where $\vec{N}$ is the outward unit normal vector,

$$\hat{p}_i(G(x)) = \frac{1}{N}\sum_{j=1}^{N} K(G(x) - G(x_j))$$

for $x_j \in R_i$ and K is a Gaussian kernel [35]. For speed concerns, $\hat{p}_i$ is approximated using the Fast Gauss Transform as implemented in the FIGTree library [49].

Using (3.20) and (3.22), the final speed function from (3.18) can be calculated as

$$F = -\mu\left[\Delta\phi - \operatorname{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right)\right] + \nu\frac{\partial C}{\partial t}. \tag{3.23}$$

Based on the results in Li et al. [39], $\nabla_x \phi$ and $\nabla_y \phi$ can be approximated by the central difference, and the time step is limited by $(\Delta t)\mu < \frac{1}{4}$. However, it has been shown in the implementation of the proposed algorithm that restricting (Δt) and μ so that $(\Delta t)\mu \approx 0.05$ yields much higher stability.

The initial value for C is chosen as the boundary of the foreground resulting from the previous local thresholding operation. Let R_1 be defined as the foreground, R_2 be the background, and $\phi(x, y, 0)$ be set as

$$\phi(x,y,0) = \begin{cases} 1, & (x,y) \in R_1; \\ 0, & (x,y) \in C; \\ -1, & (x,y) \in R_2. \end{cases}$$

As the level set algorithm is only used as a method to refine existing thresholding result, it tends to converge fairly quickly. The convergence criteria for the level set algorithm is set as a sequence of five time steps where the number of pixels whose labels changed from one step to the next is less than 0.1% of the total number of pixels. For Figure 3.2, the threshold tends to be around 250 pixels.

Select time points of the level set algorithm are shown in Figure 3.6. It can be seen that the final image is extremely noisy. The image is cleaned by setting any pixel to foreground which touches by 6 or more foreground pixels. This eliminates spurious pixels in the background as well as filling in holes in the cytoplasm. The results of the smoothing can be seen in Figure 3.7.

Now that a reasonable estimate of which pixels are considered foreground has

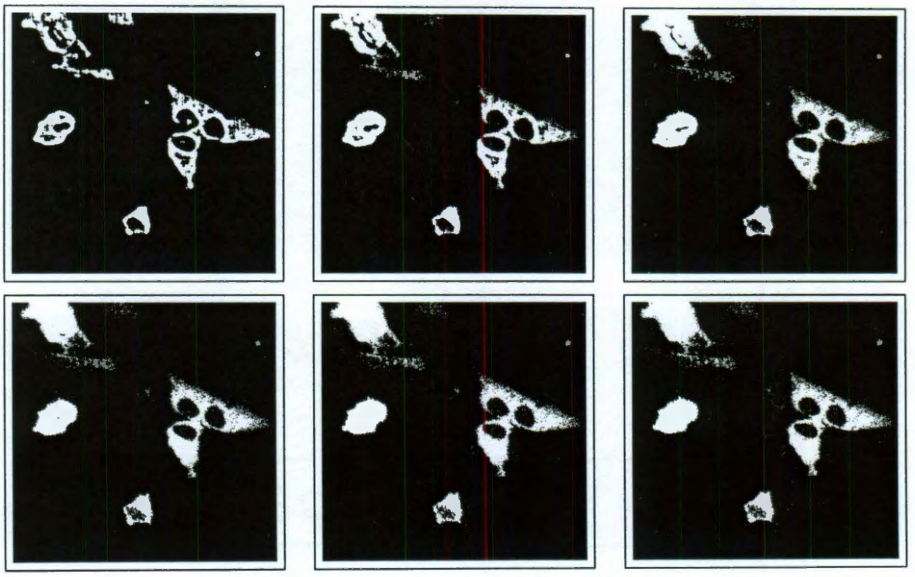


Figure 3.6 : Level set iterations for $t = 0$, 4, 8, 12, 16, and 20.

Figure 3.7 : Smoothed result from level set algorithm

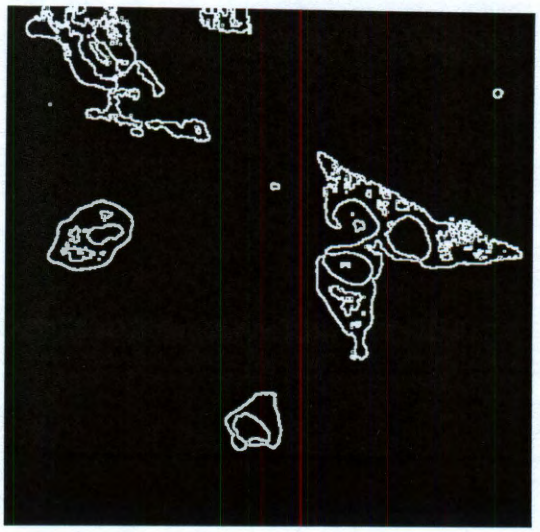


Figure 3.8 : Edge map resulting from Local Threshold (Figure 3.5)

been obtained, the final edge image needs to be determined. This is done by taking a one pixel erosion of the the threshold image, shown in Figure 3.5. A new image is generated using a pixel by pixel logical XOR operation between the original local threshold image and the eroded local threshold image. The logical XOR operation will set as foreground any pixels which are not the same in the two images, and set as background any pixels which are the same in the two images. The XOR step yields the edge image seen in Figure 3.8.

If the background is noisy, there will be some false edges present due to the background. To remove these false edges, the level set image is used as a mask. That is, if the edge pixel is near (within a certain number of pixels) of a foreground pixel in the level set result, then that edge pixel is kept. This two-stage method results in

Figure 3.9 : Edge map after masking Figure 3.8 with Figure 3.7

the edge image seen in Figure 3.9.

3.3.3 Ellipse Detection

A primary goal in computer vision is object detection and recognition. Starting with an edge image, the computer must be able to determine the location and size of different primitive shapes in an image. For example, in a child's drawing of a house, an adult can immediately recognize in the rectangular structure with a triangle for the roof. The question becomes how best to let a computer "see" these primitive objects.

The most common and basic method of shape recognition is the Hough transform. The Hough transform is a voting procedure, in which every combination of edge pixels

is tested to determine the parameters of the shape in question. For example, if the goal were to find a line in the image, then the Hough transform would begin by transforming the line into its parametric form, that is finding all values of (r, θ) which fulfill

$$x \cos\theta + y \sin\theta = r,$$

where (r, θ) is constant for every (x_i, y_i) on the line. For every edge point (x_i, y_i), the corresponding sinusoidal curve in (r, θ)-space is plotted. The intersections of the resulting sinusoidal curves would be candidate lines in the edge image. [24]

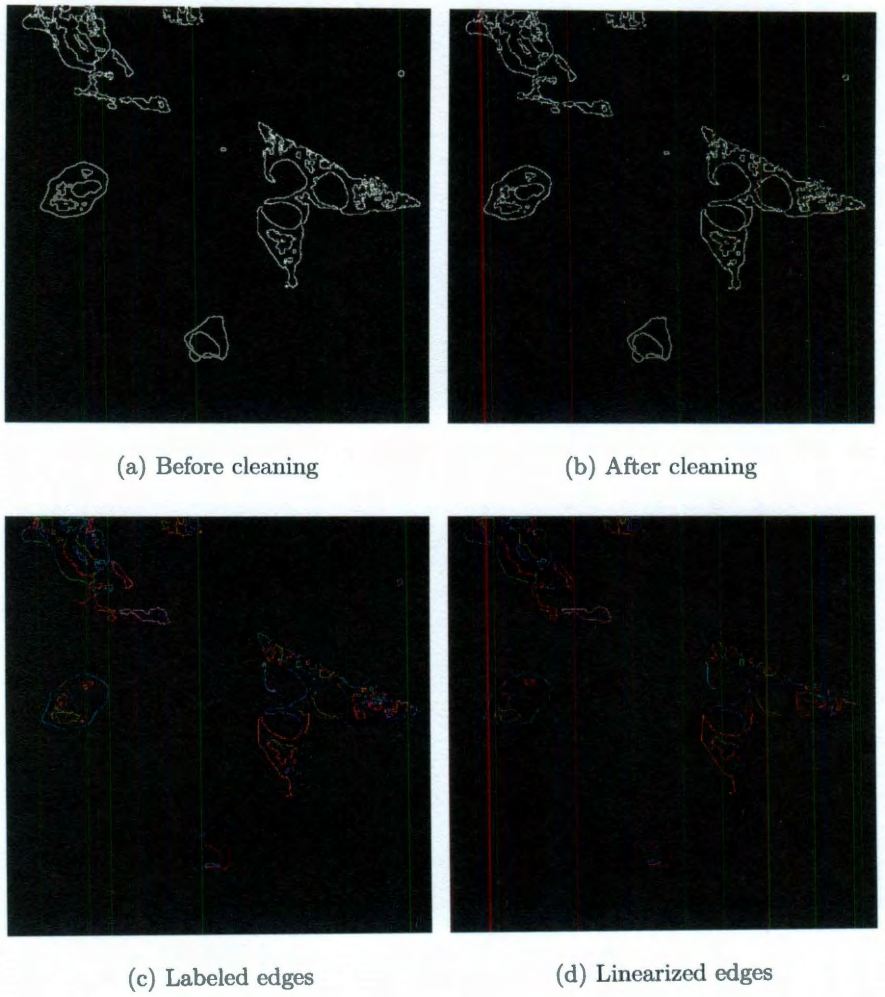
For ellipse detection, the Hough transform requires a five-dimensional accumulator for the center of the ellipse (x and y), the length of the major axis, the length of the minor axis, and the angle of rotation. This is naively done by examining every combination of five edge points in the image and calculating the parameters of the ellipse passing through those five points. As a result, the Hough transform is extremely slow and highly dependent on the accuracy of the accumulator [15]. For computational speed, the proposed algorithm uses an algorithm proposed by Nyugen et al., in which linearized edges are grouped together under certain conditions to form candidate ellipses. These ellipses are scored based on how well they fit the edge image [53].

Nyugen's algorithm begins by cleaning the edge image by removing isolated points from the edge image, as well as "over-connected" points - those points having more than two 4-connected (horizontal or vertical) neighbors. The result of the cleaning

is included as Figure 3.10b. These edges are traced and any edge with less than a given threshold length (chosen as 3 pixels in this example) is removed from the image. The cleaned and labeled edge image is included as Figure 3.10c. After the image is cleaned, the remaining edges are linearized. To linearize an edge image, each edge is iteratively fit to a series of line segments so that there is no more than some distance δ between a true edge point and the line that approximates it. The linearization algorithm for was adapted from MATLAB source code written by Peter Kovesi, in which the line segment under consideration, say AB, is split into a second line segment at the point with the highest error [36]. That is, the line segment AB is split into AC and CB where C is the point with the maximum error δ. Only the segment endpoints are kept for later analysis. The linearized edges are included as Figure 3.10d.

As can be seen in in Figure 3.10d, the linearized edges tend to run together over what is actually an ellipse edge and what is not. As a result, the edges will be broken into arcs based on three criteria. These criteria include a test for overall curvature, a test for line segment length, and a test to ensure that the angles between line segment do not differ by more than a given threshold. These conditions are summarized in Figure 3.11 (taken from [53]). Figure 3.12 shows the results of breaking the linearized edges into potential arcs.

For the curvature test, two segmentation points to either side of the point in



(a) Before cleaning

(b) After cleaning

(c) Labeled edges

(d) Linearized edges

Figure 3.10 : Initial steps in the ellipse detection algorithm

question $O_{m,n}$ are examined, and the values α_1, α_2, α_3, and α_4 are calculated by

$$\alpha_1 = \angle O_{m,n}O_{m,n-2}O_{m,n-1},$$

$$\alpha_2 = \angle O_{m,n-1}O_{m,n-2}O_{m,n+1},$$

$$\alpha_3 = \angle O_{m,n}O_{m,n+2}O_{m,n+1}, \text{ and}$$

$$\alpha_4 = \angle O_{m,n+1}O_{m,n+2}O_{m,n+1}.$$

$O_{m,n}$ is considered to be a split between two potential arcs if one of the following conditions are satisfied:

$$\alpha_1 \times \alpha_2 < 0 \text{ or}$$

$$|\alpha_2| - |\alpha_1| < 0 \text{ or}$$

$$\alpha_3 \times \alpha_4 < 0 \text{ or}$$

$$|\alpha_4| - |\alpha_3| < 0.$$

The distance test checks to make sure two consecutive line segments are not significantly different in length. Let

$$a = ||O_{m,n-1}O_{m,n}|| \qquad \text{and} \qquad b = ||O_{m,n}O_{m,n+1}||.$$

Then $O_{m,n}$ is considered a split between two potential arcs if either

$$\frac{a}{b} < \frac{1}{4} \qquad \text{or} \qquad \frac{a}{b} > 4.$$

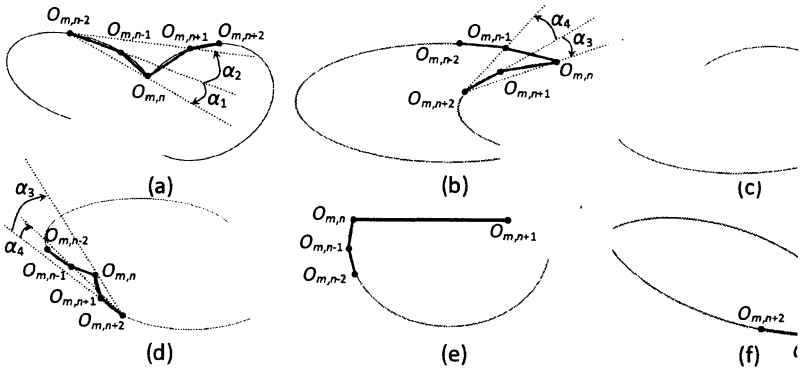


Figure 3.11 : "Curve segmentation conditions, (a): $(\alpha_1 \times \alpha_2 < 0)$, (b): $(\alpha_3 \times \alpha_4 < 0)$, (c): $(|\alpha_2| - |\alpha_1| < 0)$, (d): $(|\alpha_4| - |\alpha_3| < 0)$, (e): The length condition, (f): The angle condition" [53].

The angle test makes sure neighboring angles are roughly the same size. This test is performed by calculating β_1, β_2, and β_3 as follows:

$$\beta_1 = \angle O_{m,n-2} O_{m,n-1} O_{m,n},$$

$$\beta_2 = \angle O_{m,n-1} O_{m,n} O_{m,n+1}, \text{ and}$$

$$\beta_3 = \angle O_{m,n} O_{m,n+1} O_{m,n+2}.$$

The linearized edge is split into arcs at point $O_{m,n}$ if

$$|\beta_1| - |\beta_2| > TH_\theta \qquad \text{and} \qquad |\beta_3| - |\beta_2| > TH_\theta,$$

where TH_θ is an angle threshold set by the user. In this example, the threshold TH_θ has been set to 40°.

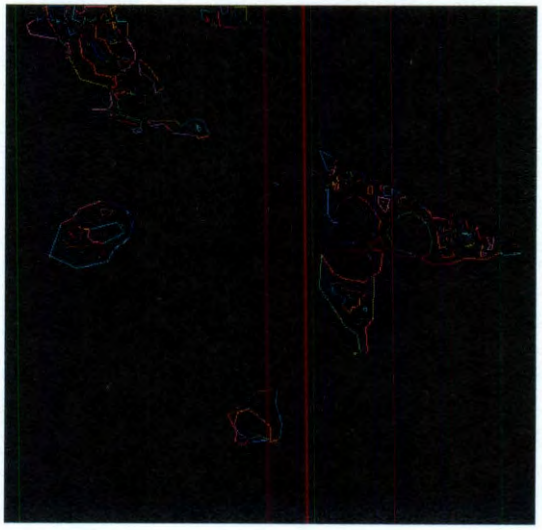


Figure 3.12 : Arcs resulting from split by curvature, length, and angle conditions

All linearized segment endpoints for which these three conditions can be checked are tested. For any point in which at least one of these conditions is fulfilled, the segment endpoints before the split point are stored as a single arc, and the testing continues with the remaning segment endpoints.

Once these K arcs have been calculated, they are grouped back together into reasonable ellipse approximations in both in a local manner and a global manner. For the local method, for each of the K arcs (m), every other arc (n) is examined to determine the minimum distance $D_{m,n}$ between the endpoints of m and n. For those arcs n where $D_{m,n} < d_0$, given the predetermined distance d_0, the arc is found for which the angle between the closest endpoints of n and m is a minimum. These two arcs are grouped and set aside as a candidate arc for an ellipse. Some of the local

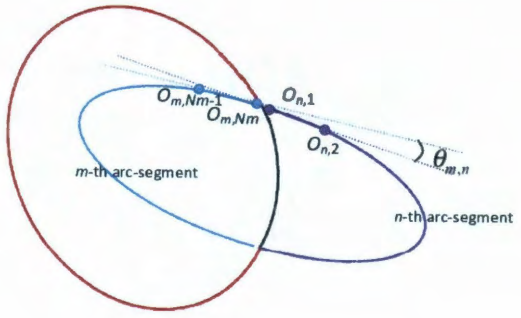


Figure 3.13 : Example of local curve grouping [53]

groupings are shown in Figure 3.13.

The global arc grouping algorithm examines each arc m and each subsequent arc n. If the minimum distance $D_{m,n}$ between the arc endpoints is less than d_0, a curvature condition is checked to make sure the two arc segments can generate a possible ellipse. This curvature condition is best explained in Figure 3.14. Let C_m and C_n be the midpoint between the two ends of segments m and n, respectively, and $O_{m,1/2}$ and $O_{n,1/2}$ be the middle segment points of the mth and nth segments, respectively. If

$$||O_{m,1/2}O_{n,1/2}|| > ||C_mO_{n,1/2}|| \quad \text{and} \quad ||O_{m,1/2}O_{n,1/2}|| > ||C_nO_{m,1/2}||$$

m and n will be grouped into a candidate arc.

Ellipses are fit to each each of these local and global arc groupings. For increased accuracy, the ellipses will be fit to the full arc segments, (all points, as opposed to just the line segment end points) according to the formula laid forth in Halir and Flusser

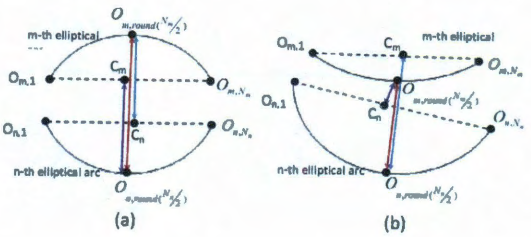


Figure 3.14 : Curvature condition for global arc grouping. (a): Pair of curves that satisfy curvature conditions, (b): Pair of curves that do not satisfy curvature conditions. [53]

[27]. This method generates coefficients for the conical equation

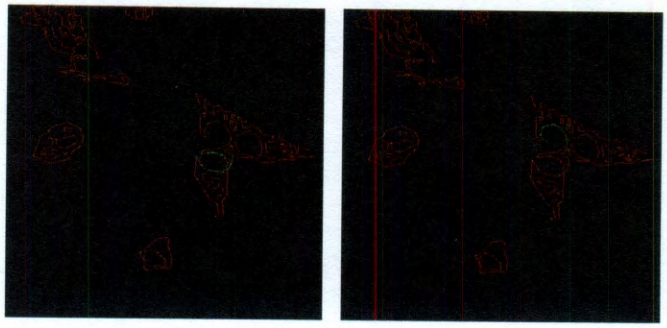
$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0, \tag{3.24}$$

which can be solved for the parameters of the standard form

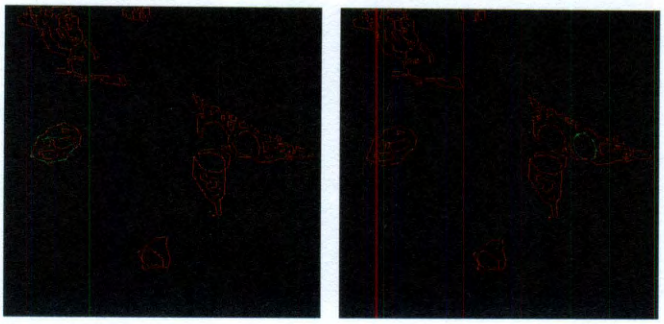
$$\frac{(x\cos\theta - y\sin\theta - O_x)^2}{R_x^2} + \frac{(y\cos\theta + x\sin\theta - O_y)^2}{R_y^2} = 1. \tag{3.25}$$

Several of the ellipses found using this method are included in Figure 3.15. As can be seen, the nuclei of the different cells have been located, as well as several errant ellipses. As a result, there must be a method to scoring the ellipses to determine which ellipses should be kept and which should be discarded.

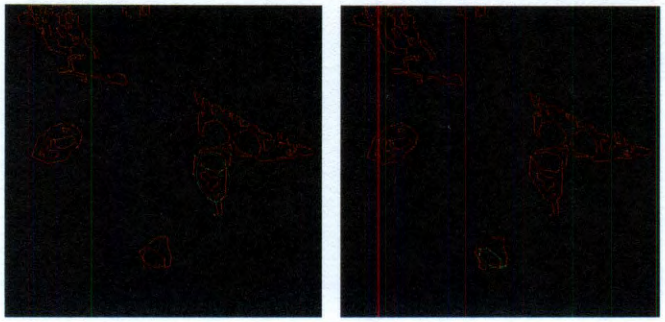
The scoring method used in the proposed algorithm is derived from Yao et al.'s genetic algorithm fitness evaluation [72]. The propsed algorithm combine two fitness measures, denoted as "Similarity" and "Distance". The Similarity fitness is calculated



(a) Global Ellipse result (b) Global Ellipse result



(c) Global Ellipse result (d) Local Ellipse result



(e) Local Ellipse result (f) Local Ellipse result

Figure 3.15 : Several located ellipses from the described algorithm. The original edge image is shown in red, with the located ellipse shown in green. The green "X"s are the segmentation points making up the arc segments the ellipse is derived from.

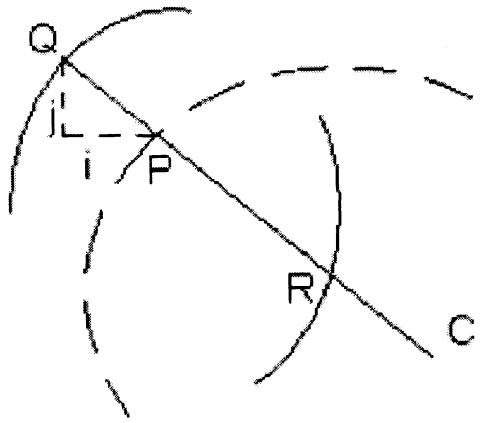


Figure 3.16 : "Matching of a candidate ellipse, point by point, to potential actual ellipses in an image [72]." Point P is a point on the ideal ellipse, Q is an actual edge point, C is the center of the ideal ellipse

as

$$S = \frac{1}{\text{Total number of points}} \sum_{(x,y)} \frac{E(x+i, y+j)}{d_{x,y}} \tag{3.26}$$

where $E(x+i, y+j)$ is an indicator function of whether there is an edge point at the coordinates $(x+i, y+j)$, where $(x+i, y+j)$ is within some distance of (x, y). The similarity is determined point-by-point based on the ideal ellipse described by the previously calculated parameters, as shown in Figure 3.16. For each point P on the ideal ellipse, the closest edge point Q that lies on the line between P and C is found, whether Q is to the inside or the outside of the ideal ellipse.

In (3.26), $d_{x,y}$ is a distance penalty given by

$$d_{x,y} = \exp\left[\frac{|i| + |j|}{4}\right] \tag{3.27}$$

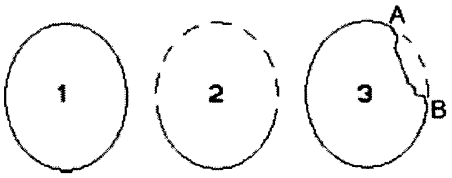


Figure 3.17 : "Perfect and imperfect ellipses [72]." Solid lines indicate edge images, and dashed lines indicate ideal ellipses

that penalizes the distance between the points P and Q as in Figure 3.16. The Distance fitness measure is calculated as

$$D = \frac{1}{\text{Effective number of points}} \sum_{(x,y)} d_{x,y} \tag{3.28}$$

where the effective number of points is the number of points on the ideal ellipse that match to an actual edge pixel, and $d_{x,y}$ is calculated as in (3.27). The difference between similarity and distance measures are depicted in Figure 3.17. Figure 3.17, Ellipse 1 is a perfect fit, with $S \approx 1$ and $D \approx 0$. Ellipse 2 is missing half of the ideal ellipse (dashed line) and has $S \approx 0.5$ and $D \approx 0$, because those edge points (solid lines) that exist match the ideal ellipse (dashed lines) exactly. Ellipse 3 has an S value between the two other ellipses, and a higher D value due to the segment AB. To score the ellipses, both the S value and the D value are calculated, with the final single score as $S/(1 + D)$.

The proposed algorithm cycles through the ellipses to determine which are good potential nuclei and which are extraneous detected ellipses due only to the current

shape and position of boundaries. There are three primary methods of determining which ellipses should be kept. First, the ellipses are examined in order of best score to worst. For each ellipse, if the ellipse overlaps any previously confirmed ellipse, it is discarded. Second, if the percentage of theoretical edge pixels matched failes to fall above a defined threshold, the ellipse is discarded. Finally, if the relationship between the average intensity of pixels inside the ellipse is and the average intensity of pixels in a small neighborhood outside the ellipse is not correct, then the ellipse is discarded.

As can be seen in Figure 3.18, the results are not perfect. There are several extraneous ellipses detected in the upper left corner of the image. In addition, the nuclear boundary for the cell in the lower center of the image is not kept. However, the detected nuclei are sufficient for the next phase of the initial segmentation: the detection of the cytoplasmic boundaries.

3.3.4 Cytoplasm Boundary Detection

Once the locations of the nuclei have been estimated, it is possible to determine the boundaries of the appropriate cytoplasm region for each cell. The proposed algorithm uses an implementation of an algorithm by Jones et al. [33]. To detect the cytoplasmic boundaries, the presence of foreground "blobs" – large groups of connected foreground pixels – are found in the level set image. One or more of the calculated ellipses are associated with each blob. This is accomplished by determining whether the center of

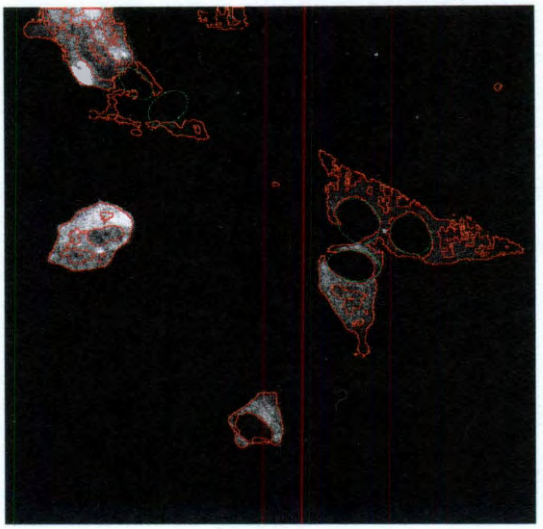


Figure 3.18 : Overlay of highest scoring ellipses with the original image.

the calculated ellipse lies within the smallest rectangle containing all of the foreground blob. If there are more than one ellipse associated with any given blob, then Djikstra's Algorithm is used to assign each pixel in that blob to the "nearest" ellipse, with all pixels in each nuclei having distance value 0. The distance metric D is taken from Jones et al. [33] and is given by

$$D(d\boldsymbol{x})^2 = \frac{(d\boldsymbol{x}^T \nabla \mathcal{G})^2 + \lambda (d\boldsymbol{x}^T d\boldsymbol{x})^2}{\lambda + 1}, \tag{3.29}$$

where $\nabla \mathcal{G}$ is the appropriate gradient of a Gaussian-smoothed version of the image, and λ is a weighting parameter based on the difference between the mean foreground and background pixel intensities.

Figure 3.19 is a translucent overlay of the detected nuclear and cytoplasmic regions onto the original image (Figure 3.2). As can be seen, the proposed algorithm

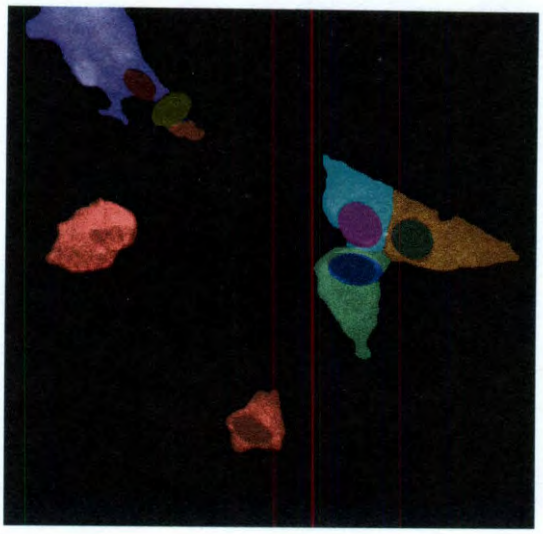


Figure 3.19 : Overlay of the detected nuclear and cellular boundaries on the original image

obtains accurate boundaries of the cytoplasmic areas. The nuclear estimation portion, however, yields results which appear to be a few pixels off in most cases.

3.4 Proposed Algorithm

The proposed algorithm is a modification of the algorithm originally used in Cell Tracker. Under the assumption that the classification of every pixel in a given frame is known, the algorithm predicts the classification of all the pixels in each subsequent frame. For this chapter, the current frame will be indicative of the frame being processed, and the previous frame will be indicative of the frame which has just been completed.

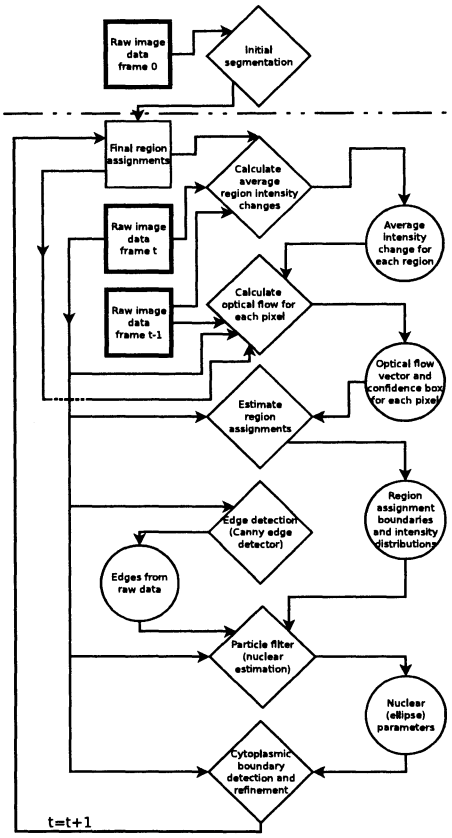


Figure 3.1 : A flowchart for the proposed tracking algorithm. Diamond entries are calculation and estimation steps. Square entries are considered permanent data - i.e. data provided by the user or data is output by the program. Round entries are temporary data - i.e. data which is only relevent for calculation of a subsequent step in the current frame.

Recall Figure 3.1. With the exception of Chapter 3.4.1, each of the following sections refers to one of the diamond entries under the horizontal line.

The first step in the tracking algorithm is reduction of noise in the current frame using spatial and temporal smoothing methods. These methods are described in detail in Chapter 3.4.1. Once the current frame has been smoothed, the average change in the mean intensity for each distinct region from the previous to the current frame is calculated. Recall that the regions in each frame consist of background, each cellular compartment, and one region consisting of all the unidentified foreground objects. The average change is calculated using a slight modification of the optical flow algorithm given by Haussecker and Fleet as described in Chapter 3.1.3. The modifications of the Haussecker and Fleet optical flow algorithm and other pertinant details are discussed in Chapter 3.4.2. A modification of the Lucas-Kanade optical flow method (Chapter 3.1.2) is combined with the average brightness change for each region to calculate the optical flow for each pixel in the previous frame (details in Chapter 3.4.3). In addition to the direct optical flow vector, a 95% confidence interval is calculated for each component of the optical flow vector. This 95% confidence interval is used to estimate the region assignment for each pixel in the current frame. The details for this estimation are given in Chapter 3.4.4. The estimated classification is cleaned through the use of a block voting scheme, which is discussed in Chapter 3.4.5.

Due to potential inaccuracies in the optical flow calculation and classification

estimates, the optical flow vector field alone is unsuitable for tracking. The proposed algorithm combines the classification estimates with intensity data from the current frame to obtain a more accurate estimation of the nuclear and cytoplasmic boundaries. To estimate the nuclear ellipse parameters, the Particle Filter algorithm is used, with the specifics given in Chapter 3.4.6. A global threshold is calculated for the frame, and the resulting binary image is used to determine the cytoplasmic boundaries. The method used to calculate this threshold and the resulting cytoplasmic segmentation are described in Chapter 3.4.7

This chapter is illustrated by tracking the regions from Figure 3.19 (the previous frame) into Figure 3.20 (the current frame). Some intermediate and the final results for tracking the current frame are shown in Figure 3.26.

3.4.1 Image Smoothing

For each optical flow calculation, every frame is first smoothed with respect to both space and time. This is accomplished by first using a basic smoothing algorithm over a square spatial neighborhood of each pixel. The edges in the frame are not as important to the calculation of optical flow as they are to the initial segmentation, so the the user is given a choice between Gaussian smoothing (Gaussian kernel), Mean smoothing (the average of a uniform kernel), and Median smoothing (the median of a uniform kernel). If the square neighborhood extends beyond the edges of the frame, the frame is supplemented by using the nearest known value for each missing point.

Figure 3.20 : Raw data for "current frame" (frame 1) used as a reference for Chapter 3.4.

After all the frames have been spatially smoothed, the algorithm smooths the previous frame with respect to time. For each pixel location, the mean, median, or Gaussian-weighted mean of that location is calculated with a radius in time. If the time radius extends beyond the first or last frame in the sequence, the data is supplemented by repeating the pixel value from the first or last frame in the time series as appropriate.

3.4.2 Prediction of Region Mean Intensity

A modification of Haussecker and Fleet's algorithm is employed to detect the mean intensity change in each region in the previous frame. Recall from Chapter 3.1.3 that

the undifferentiated brightness change equation is given by

$$I(\boldsymbol{x}(t), t) = h(g_0, t, \boldsymbol{a}), \tag{3.30}$$

where g_0 is the intensity in the initial frame and $\boldsymbol{x}(t)$ is a path that a pixel takes through the time series. Haussecker and Fleet are able to use this form because they calculate an underlying model h for the entire time series. If the underlying model of region brightness change was known for the time series being examined, (3.30) would be sufficient. However, as the goal is to estimate the underlying model, the brightness change must be calculated piecewise.

For the remainder of this section, the current frame is the image indexed at t, and the previous frame is the image indexed at $t-1$.

For the piecewise-defined model, the brightness change model is defined to be dependent only on the previous frame. Thus,

$$I(\boldsymbol{x}(t), t) = h(g_{t-1}, \boldsymbol{a}). \tag{3.31}$$

With this in mind, a simple linear model is chosen for h to prevent the presence of a severely underdetermined system. In fact, h is chosen to be

$$h(g_{t-1}, \boldsymbol{a}) = a_0 g_{t-1} + a_1.$$

Let R be a defined region of the image. Regions in each image include the cytoplasms and nuclei of each cell and the background, as well as unidentified foreground objects. The pixel intensities in each of the r regions are assumed to be normally

distributed as

$$I(\boldsymbol{x}(t), t-1) \sim \sum_{i=1}^{r} 1_{\boldsymbol{x}(t)\in R_i} N\left(\mu_{R_i}, \sigma_{R_i}^2\right).$$

The goal of this stage of the algorithm is to predict the shift in the intensity distribution mean $\Delta\mu_{R_i}$ for all i, as opposed to the shift in each pixel path intensity difference $\Delta I(\boldsymbol{x}(t))$. However, the obvious way to predict $\Delta\mu_{R_i}$ is by the average of $\Delta I(\boldsymbol{x}(t))$ for all $\boldsymbol{x}(t) \in R_i$. For computational simplicity and speed, a reasonable sample of the pixels in R_i will be used as opposed to all of the pixels in the region.

The algorithm is as follows: Let M be the number of pixels to be sampled from region R_i, with N being the number of regions in the image (including the background). To calculate the average region intensity change, select M pixels p_m from each region R_i such that the square neighborhood of pixels around p_m is entirely contained within R_i. Now, for each p_j, Haussecker and Fleet's algorithm (given by (3.11)-(3.15)) is used. For these equations, f is defined as the Taylor series given by

$$f(g_{t-1}, \boldsymbol{a}) = a_0 g_{t-1}.$$

$\bar{f}(R_i)$ is calculated as the average of

$$\bar{f}(R_i, t-1) = \frac{1}{M}\sum_{m=1}^{M}(a_0)_m(g_{t-1})_m$$

for each region i.

The optical flow vectors calculated by Haussecker and Fleet's algorithm are not kept for later use, because the optical flow result is highly dependent on the estimated brightness change of the pixel. However, as goal of this calculation is the average

brightness change for each region to be used in calculating the optical flow for all the pixels in the region, the difference between the optical flow vectors calculated from the exact brightness change for a pixel and the average brightness change for the region would cause errors in later stages of the tracking algorithm.

It is known that estimating a nonlinear function by piecewise linear functions is a rough estimate. As a result, the forward difference of the piecewise linear estimate is smoothed using simple exponential smoothing for $t > 1$, allowing

$$f^*(R_i, t-1) = \alpha * \bar{f}(R_i, t-1) + (1-\alpha) f^*(R_i, t-2), \tag{3.32}$$

where α is a weight between 0 and 1 chosen by the user.

3.4.3 Optical Flow Calculation

To actually predict the optical flow, a slight modification of the Lucas-Kanade method will be used in the interests of speed and because it is a least squares method, for which the error of the estimate can easily be calculated. Recall that the optical flow calculation is for the previous frame, and is a measurement of the movement of objects from their positions in the previous frame to their positions in the current frame.

The brightness constraint equation (3.1) is modified to become

$$\frac{\partial}{\partial x} I(x,y)\boldsymbol{v}_x(x,y) + \frac{\partial}{\partial y} I(x,y)\boldsymbol{v}_y(x,y) + \frac{\partial}{\partial t} I(x,y) = \sum_{i=1}^{N} 1_{(x,y)\in R_i} f^*(R_i, t), \tag{3.33}$$

where there are N regions R_i in the previous frame, and f^* as defined in (3.32). By drawing from the Lucas-Kanade equation as given in (3.7), the calculation of optical

flow is the solution to the least squares problem for each pixel p

$$\begin{pmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} -I_t(q_1) + 1_{q_1 \in R_i} \sum_{i=1}^{N} f^*(R_i, t) \\ -I_t(q_2) + 1_{q_2 \in R_i} \sum_{i=1}^{N} f^*(R_i, t) \\ \vdots \end{pmatrix}, \tag{3.34}$$

where the q_k are the k pixels surrounding and including p. (3.34) is solved via least squares using the Jacobi Singular Value Decomposition (SVD) algorithm implemented in the Eigen library [31] to minimize the effects of numerical instability.

To determine the least squares error, the residuals to the least squares fit are calculated as

$$\boldsymbol{r} = \boldsymbol{Q}\boldsymbol{v} - \boldsymbol{t},$$

where $\boldsymbol{r}$ is the vector of residuals, $\boldsymbol{Q}$ is the $n \times 2$ gradient of the image, and $\boldsymbol{t}$ is the left-hand side of (3.34). The mean squared error of the residuals is calculated as

$$MSE = \frac{\boldsymbol{r}^T\boldsymbol{r}}{k-2},$$

By the properties of the SVD,

$$\begin{aligned} \boldsymbol{Q}^T\boldsymbol{Q} &= (\boldsymbol{U}\boldsymbol{S}\boldsymbol{V})^T(\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}) \\ &= \boldsymbol{V}^T\boldsymbol{S}^T\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{S}\boldsymbol{V} \\ &= \boldsymbol{V}^T\boldsymbol{S}^2\boldsymbol{V}, \end{aligned}$$

because $\boldsymbol{U}^T\boldsymbol{U}$ is the identity matrix, and $\boldsymbol{S}$ is a diagonal matrix. This can be used

to calculate the standard deviation of $\boldsymbol{v}$ as

$$\sigma^2(\boldsymbol{v}_x(p)) = MSE * (\boldsymbol{Q}^T\boldsymbol{Q})^{-1}[0,0] \tag{3.35}$$

$$\sigma^2(\boldsymbol{v}_y(p)) = MSE * (\boldsymbol{Q}^T\boldsymbol{Q})^{-1}[1,1], \tag{3.36}$$

which yields the 95% confidence intervals for both $\boldsymbol{v}_x$ and $\boldsymbol{v}_y$.

3.4.4 Initial Pixel Assignment

The initial pixel assignment follows from the 95% confidence intervals of the optical flow calculated in Chapter 3.4.3. Using these confidence intervals, a list of the possible assignments for every pixel is built. First, assume that pixel p is in region R_i, which will be referred to by its label i (i.e. the label of p is i). For any pixel q in the rectangular confidence region of the optical flow for pixel p, q is marked as potentially associated with p.

There may be some pixels in the previous frame for which the optical flow could not be calculated due to either numerical instability or singular matrices. These pixels are detected by one of two conditions. The first condition is that the distance between the mean of the optical flow estimate and the intersections of the lower 95% confidence bounds is larger than 100. The second condition is that the lower confidence bound in either the x or the y direction is greater than $1,000$. If either of these cases occur for pixel p with label i, the optical flow for that pixel is assumed to be $\boldsymbol{v}(x,y) = (0,0)$ with confidence intervals of $(\pm X)$ in both the x and y direction, where X is defined by the user.

Single and multiple associations

After all of the associations have been made, the algorithm cycles through all of pixels in the current frame. If a pixel q in the current frame has only a single association with a pixel p in the previous frame, then the assignment of q becomes the the assignment of p in the previous frame.

If a point has multiple associations, then the region assignment will need to be estimated. Assume q is the pixel in the current frame t being analyzed. Let p be the pixel in the previous frame $t-1$ associated with region R_i for some i. The likelihood of q being the next location of p is given by

$$L(p \to q) = f_1(I(p,t-1) + f^*(R_i,t-1) - I(q,t)) \cdot f_2(||q - (p + \boldsymbol{v}(p,t-1))||), \quad (3.37)$$

where p is assigned to region i, and f^* is calculated as in (3.32). It is reasonable to assume that f_1 is the density function of a Normal distribution and f_2 is the density function of an Exponential distribution. Thus, f_1 is calculating the likelihood of seeing the difference between the observed intensity at pixel q and the expected intensity of p under a Normal distribution with mean 0 and a variance given by the user. Similarly, f_2 is calculating the likelihood of seeing the distance between the location of pixel q and the expected location of pixel p given the optical flow from frame $t-1$ under an Exponential distribution with scale parameter given by the user.

To determine the assignment of q, the total likelihood of q belonging to region i

is calculated as

$$L(q \in R_i) = \sum_{k=1}^{K} 1_{p_k \in R_i} L(p_k \rightarrow q) \tag{3.38}$$

for each of the K pixels associated with q. Pixel q is assigned the label i of the region with the highest total likelihood.

No Associations

For pixels q which are not associated with any pixel p in the previous frame, a nearest-neighbor algorithm is used to generate associations with pixels s in the current frame which have already been assigned to regions. Using the approximate nearest neighbor library [50], a number of nearest assigned neighbors in the current frame are determined. The likelihood for these associations is given as

$$L(q \in R_i \mid p \in R_i) = f_1(I(p,t) - I(q,t)) \cdot f_2(||q - p||),$$

where f_1 is a Normal density and f_2 is an Exponential density. As in the case of multiple association, the total likelihood of q belonging to region i is given by

$$L(q \in R_i) = \sum_{k=1}^{K} 1_{p_k \in R_i} L(p_k \rightarrow q),$$

and q is assigned to the region with the highest total likelihood. The results of the initial region estimation can be seen in Figure 3.21.

3.4.5 Pixel Reclassification by Block Voting

Figure 3.21 shows the initial classification by Chapter 3.4.4. As can be seen, the initial classification results in a highly disjoint image, with significant bleed between different

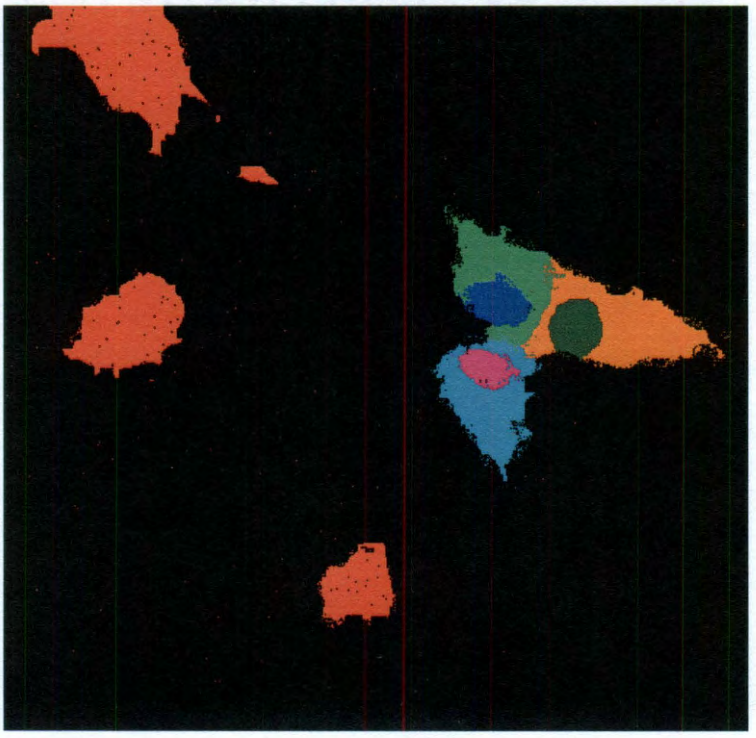


Figure 3.21 : Initial predicted regions for frame 1 of experimental data from Chapter 5.2.2 after applying the proposed algorithm through Chapter 3.4.4.

regions. The initial classification image must be cleaned, because later portions of the algorithm rely on the boundaries of this image. The image is cleaned by a block voting reclassification scheme. The block voting scheme is implemented by examining a square neighborhood around each pixel, where the center pixel is given the assignment appearing most within the square neighborhood. The block voting scheme results in the pixel classification shown in Figure 3.22

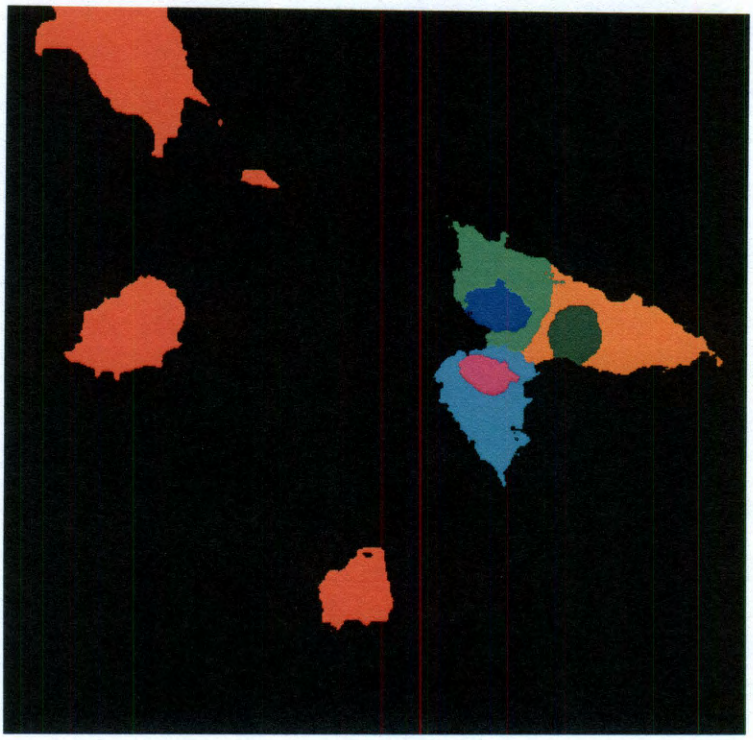


Figure 3.22 : Cleaned predicted regions for frame 1 of experimental data from Chapter 5.2.2. Cleaning was done by block voting with a one pixel radius

Canny Edge Detection

The Canny edge detector has five different steps. The first step is a smoothing of the image through a Gaussian kernel. The implementation of the Canny edge detector for the proposed algorithm accepts the kernel bandwidth σ and generates a square neighborhood of radius 3σ. The second step in the Canny edge detector is the calculation of the gradient magnitude and direction. The proposed algorithm calculates the gradient using the Sobel operators, which uses the following 3×3 structures.

$$\boldsymbol{S}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad \boldsymbol{S}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Once the magnitude and direction of the gradient have been calculated, two edge images are formed. The first is generated by thresholding the gradient magnitude with a high threshold value in an effort to limit the number of extraneous edges. At the same time, a nonmaximal suppression step is included in the formation of the edge image. In this step, nonmaximal gradient magnitudes along the direction of the gradient are discounted as potential edge pixels. The second edge image is generated like the first, except that a lower threshold value is used in an effort to include a number of reasonable edge points.

The final step in the Canny edge detector generates what is known as a hysteresis map. The hysteresis map links edge points from the high threshold edge map. The linking is accomplished through the use of edge points from the low threshold edge

map which are connected to edge points from the high threshold edge map.

3.4.6 Nuclear Estimation

Due to the high amount of noise in the data despite the smoothing performed in Chapter 3.4.1, the pixel classifications obtained through Chapter 3.4.5 do not necessarily line up well with the data. This can be seen directly in Figure 3.23, where the errors are especially noticable in the lower cell of the 3-cell cluster.

In addition, when using optical flow alone, the nuclear area occasionally either expands to cover the entire cytoplasm or shrinks to nothing in frames where there is little difference in intensities between neighboring regions, as can be seen in the tracking progression shown in Figure 3.24.

To prevent this occurence, prior knowledge is used to enhance the tracking of nuclear regions. As was discussed in Chapter 3.3, ellipses are used as a representation of nuclear boundaries. With this in mind, a variation on the Particle Filter algorithm is used to estimate the five parameters which will describe each ellipse.

In Cell Tracker, the particle filter algorithm is implemented as a selection of several hundred random ellipses under certain deviations. These random ellipses are compared to edges detected in the data by testing lines normal to the ellipse along M points on the boundary of the ellipse. For each of these ellipses, the likelihood of the ellipse being in its true location is calculated as

$$p \propto 1 + \frac{1}{\sqrt{2\pi}\sigma\lambda} \sum_{m=1}^{M} e^{-f(\mu^2, d_m^2)/2\sigma^2}, \tag{3.39}$$

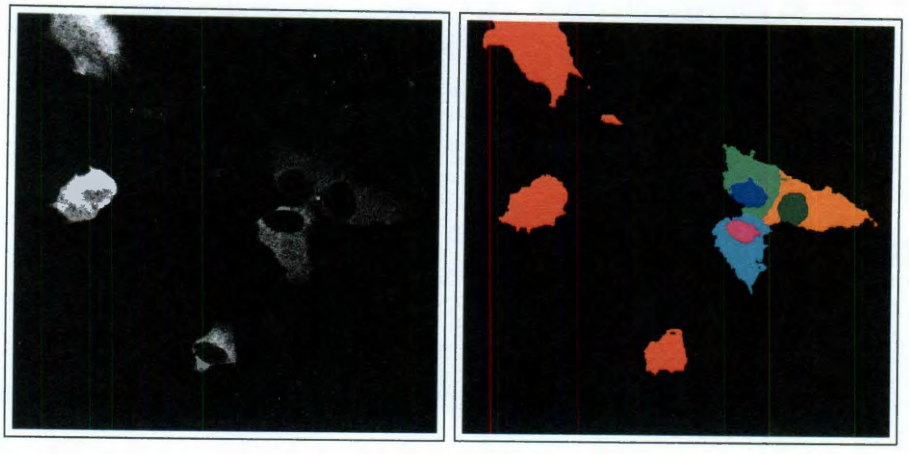


Figure 3.23 : A comparison of the cleaned optical flow regions (Chapter 3.4.5) with the original data from the current frame.

where μ is the maximum distance allowed, d_m is the distance between a point q on the edge of the ellipse and the nearest image edge point along a line normal to the ellipse at q, and λ is a scaling parameter. Cell Tracker defines $f(\mu^2, d_m^2)$ to be the minimum of μ^2 and d_m^2. The estimated ellipse is set to be the expected value of the empirical distribution of the generated ellipses. [61]

The proposed algorithm extends and modifies Cell Tracker's implementation for the scoring scheme. However, instead of relying solely on the edges obtained from the Canny edge detector, raw image data, the boundaries between the assigned regions from Chapters 3.4.3-3.4.5 are also taken into account. In addition, the proposed algorithm uses the predicted nuclear intensities from Chapter 3.4.2. Both the detected edges from the current frame and the estimated region boundaries for the current

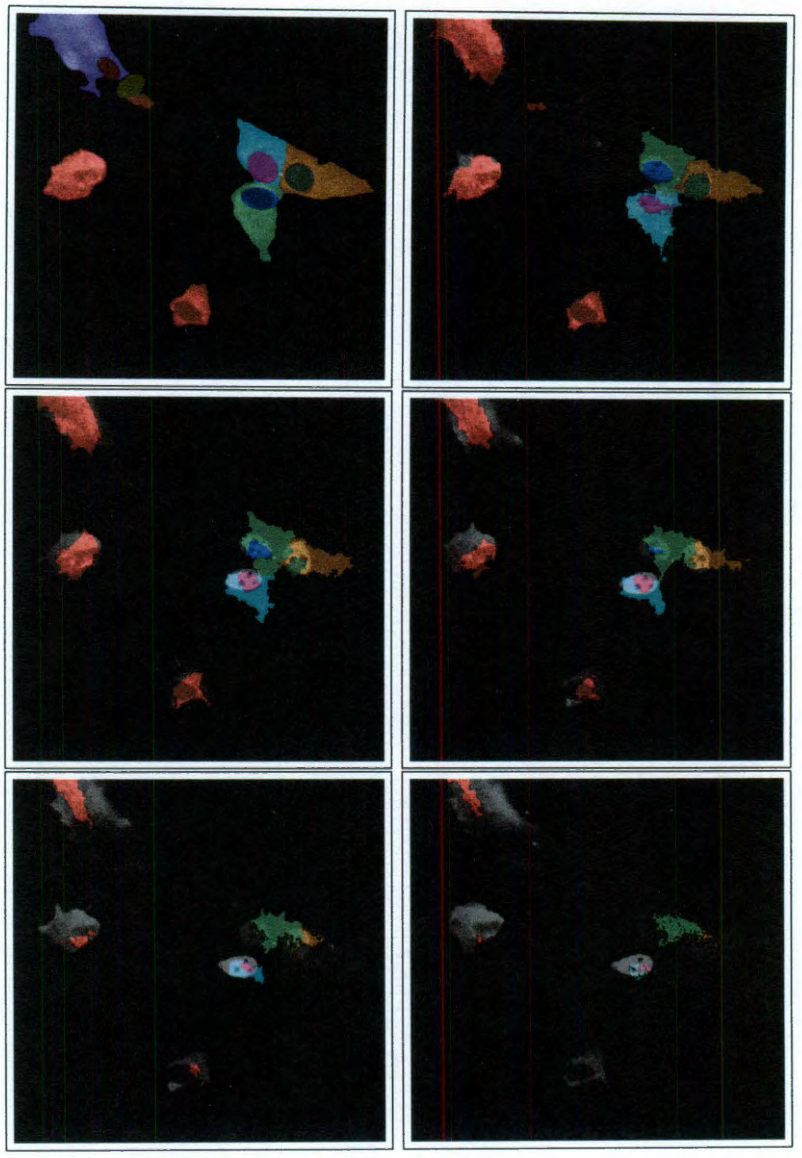


Figure 3.24 : The effects of only tracking by optical flow. For this example, the proposed algorithm was stopped after Chapter 3.4.5 in each iteration. The boundaries of both the cytoplasm and the nucleus dwindle away to almost nothing by the end of the 11th frame.

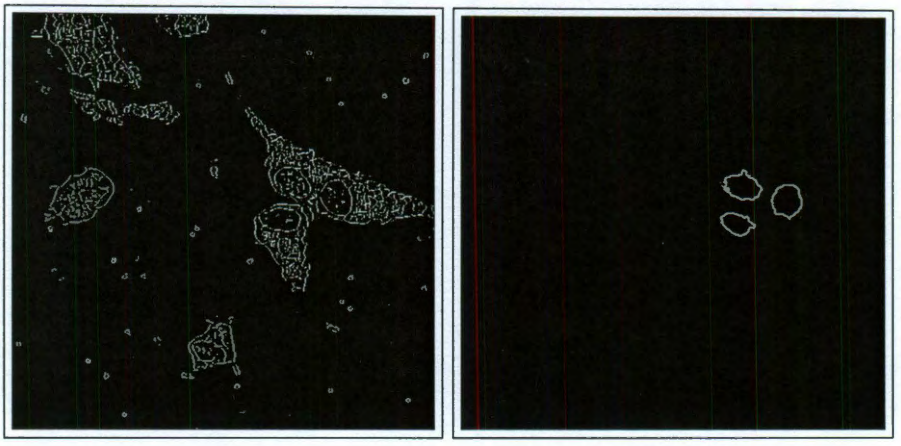


Figure 3.25 : Edge images used for the particle filter algorithm. The edge image on the left is the result of using the Canny Edge detector with an automated upper threshold as the second tertile boundary and the lower threshold as roughly the first quartile boundary of the intensity gradient magnitude values. The edge image on the right arises from the nuclear region boundaries from Figure 3.22.

frame are shown in Figure 3.25. It is clear the edges obtained from the raw data are useful - when they are available. For that reason, the proposed algorithm was designed to transfer smoothly from using the detected edges from the current frame to using the boundaries of the assigned regions as the nuclear and cytoplasmic intensity distributions converge.

Weighting Between Edges from Raw Data and Edges from Assigned Regions

The pixel intensities in both the nuclear and cytoplasmic regions for each cell are assumed to be normally distributed with different means and variances for each region. Let $\hat{\mu}_N$ be the calculated mean of the nuclear pixel intensities and $\hat{\mu}_C$ be the corresponding mean of the cytoplasmic intensities. Also let $\hat{\sigma}_C^2$ and $\hat{\sigma}_N^2$ be the calculated variance of the cytoplasmic and nuclear pixel intensities, respectively. The squared Hellinger distance will be used as the basis for the weighting function. The Hellinger distance for two parametric densities is defined in [54] as

$$H^2(P,Q) = \frac{1}{2}\int \left(\sqrt{f(x)} - \sqrt{g(x)}\right)^2 dx. \tag{3.40}$$

If $\mathcal{C}$ is defined to be the distribution of cytoplasmic intensites, such that $\mathcal{C}$ is a $N(\mu_C, \sigma_C^2)$ distribution and $\mathcal{N}$ is defined to be the distribution of nuclear intensities with $\mathcal{N}$ as a $N(\mu_N, \sigma_N^2)$ distribution, then the squared Hellinger distance $H^2(\mathcal{C},\mathcal{N})$ is given by [66] as

$$H^2(\mathcal{C},\mathcal{N}) = 1 - \sqrt{\frac{2\hat{\sigma}_N\hat{\sigma}_C}{\hat{\sigma}_N^2 + \hat{\sigma}_C^2}}\, e^{-\frac{1}{4}\frac{(\hat{\mu}_N - \hat{\mu}_C)^2}{\hat{\sigma}_N^2 + \hat{\sigma}_C^2}}. \tag{3.41}$$

The Hellinger distance is a reasonable weighting function to use because the nuclear and cytoplasmic intensity distributions tend to vary uniformly over time. In addition, it is rare to observe (with the exception of nucleoli) intensity distributions with significant overlap solely due to intensity variance within each region.

Because $H^2(\mathcal{C},\mathcal{N})$, is limited to the range from 0 to 1 inclusive [54], input from

the estimated assignments will always be used. As a goal of the proposed algorithm is to only use these assignments to calculate the nuclear ellipse parameters when absolutely necessary, the weighting function ξ will be defined as

$$\xi = \min\left[\alpha H^2(\mathcal{C}, \mathcal{N}), 1\right], \tag{3.42}$$

where α is a user-defined parameter denoting the mistrust in the optical flow calculation ability. A high value of α will use the optical flow edges only when absolutely necessary, and the edges from the real data in all other frames. Similarly, a value of α less than one would indicate that the edges from the optical flow calculation should be used in nearly every frame, even when the real edges from the data are present and clearly defined.

Particle Filter

The proposed algorithm implements the particle filter algorithm, to generate a large number of random parameterized ellipses. The user is allowed to set the number of ellipses generated (Υ), the maximum distance the center of the ellipse is allowed to vary, the maximum percent change in the area of the ellipse, and the maximum change in the rotation in degrees. The default center of the the ellipses to be calculated is set to be the calculated centroid of the corresponding nuclear region from the estimated initial pixel association (Chapter 3.4.5). To move the ellipse, a uniform random distance greater than 0 and less than the user defined maximum and a uniform random direction in the range of $[0, 2\pi)$ are generated. The center point of the ellipse

is moved from the center of the assigned pixel region in the current frame according to the generated polar vector.

To vary the size of the ellipse, a uniform random number u in the range of

$$[1 - \xi x, 1 + \xi x]$$

is generated where x is the parameter input by the user and ξ is the weighting function from (3.42). The random number u will give the overall change of the ellipse area. The aspect ratio of the ellipse is changed by multiplying the major axis half-length by a random number in the range

$$[1 - 0.5\xi x, 1 + 0.5\xi x].$$

The new minor axis half-length Ry' is calculated as

$$Ry' = \frac{u \cdot Rx \cdot Ry}{Rx'}, \tag{3.43}$$

where Rx and Ry are the old major and minor axis half-lengths, respectively, and Rx' is the new major axis half-length. Including the weighting function limits the change in both the major and minor axis directions to avoid long, thin ellipses which match only sections of the cytoplasmic boundaries. The rotation of the ellipse is modified in a similar manner.

An overall score has been derived which relies primarily on how well the random ellipse fits the edges detected in the current frame. However, as the intensity distributions converge, relying only on the edge score can have the effect of drawing

the nuclear ellipse out of position either to the cytoplasmic boundary or to spurious edges within the cell. As a result, the proposed algorithm incorporates the weighting function ξ to smoothly incorporate two other scores into the total ellipse score. These scores include a score based on the average intensity of the inside of each ellipse and the assigned regions which contain portions of the ellipse.

Edge Score

The primary scoring mechanism is dependent on the distance between a point on the candidate ellipse and the nearest candidate edge point q in either the edge image calculated from the current frame or the assigned region boundaries as shown in Figure 3.25. If μ is defined to be the maximum distance allowed and d_m is defined as the distance between q and the nearest edge point as defined previously, then the edge score for an ellipse is defined as

$$S_{E_\theta}(e) \equiv \prod_{m=1}^{M} \frac{1}{\sqrt{2\pi}\frac{\mu}{6}} e^{-\frac{1}{2}\left(\frac{6}{\mu}\right)^2 \min\{\mu^2,(d_m)_\theta^2\}}, \tag{3.44}$$

where M is the number of points for which a normals from that point are examined. Also, θ is a dummy variable indicating whether the score is being calculated from edges detected in the current frame or the boundaries of the assignments, and e is the candidate ellipse.

Now, to vary smoothly between the two edge images, the following are defined:

$$S_{E_{rd}}(e) = \xi \prod_{m=1}^{M} \frac{1}{\sqrt{2\pi}\frac{\mu}{6}} e^{-\frac{1}{2}\left(\frac{6}{\mu}\right)^2 \min\{\mu^2, (d_m)_{rd}^2\}}, \text{ and} \tag{3.45}$$

$$S_{E_a}(e) = (1-\xi) \prod_{m=1}^{M} \frac{1}{\sqrt{2\pi}\frac{\mu}{6}} e^{-\frac{1}{2}\left(\frac{6}{\mu}\right)^2 \min\{\mu^2, (d_m)_a^2\}}, \text{ and} \tag{3.46}$$

$$S_E(e) = \max\{E_a(e), E_{rd}(e)\}, \tag{3.47}$$

where the rd subscript denotes the score with respect to the detected edges, a denotes the score with respect to the assignments, and ξ is given in (3.42).

Intensity score

To assist in preventing incorrect nuclear drift when the nuclear edges are not clearly visible, each ellipse is also scored base on the average intensity of its pixels. The errors in the intensity are assumed to be zero-mean Normal variates. The intensity score is calculated as the likelihood of seeing the average intensity value given the predicted average intensity of the region and a set variance. For simplicity, the variance σ^2 has been set to be 1/20th of the maximum possible value in the current frame. The final intensity score is given by

$$S_I(e) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(\bar{I} - \hat{I})^2\right\}, \tag{3.48}$$

where $\bar{I}$ is the average intensity of the ellipse e and $\hat{I}$ is the predicted average intensity of the nuclear region being approximated.

Assignment score

In addition to the intensity score, the ellipse is also scored based on how well it matches the assignments from the optical flow calculation. This score helps prevent the tracking ellipse from moving outside the cell. Let ellipse e contain K points $\{q_1, \ldots, q_K\}$, and be represented by assignment i from Chapter 3.4.5, with the corresponding cytoplasmic region assigned as h. The assignment score is defined by

$$S_A(e) = \frac{1}{K} \sum_{k=1}^{K} \left[1_{q_k \simeq i} + 1_{q_k \simeq h} \right], \tag{3.49}$$

where the $p \simeq j$ indicates that pixel p has assignment j.

Total score

For each ellipse e, the total score is calculated via (3.47), (3.48), and (3.49). In brief, the total score is given by

$$S(e) = S_E(e) \cdot S_A(e)^{1-\xi} \cdot S_I(e)^{1-\xi}. \tag{3.50}$$

The exponent values of ξ from (3.42) are included to smoothly incorporate the assignment and intensity scores when necessary.

The final parameterization of each nucleus is taken to be the weighted average of each of the five parameters, with the weight defined by the total ellipse score (3.50).

3.4.7 Cytoplasmic Refinement

The procedure used in Chapter 3.3.4 is used to estimate the cytoplasmic boundaries. However, this procedure requires the cytoplasms of the cells to be foreground items.

There is no attempt to follow the full protocol established in Chapter 3.3 as it is not necessary to distinguish nuclear edges. In the interests of speed, all that is necessary is the calculation of a reasonable global threshold in an effort to distinguish between foreground (cell and nuclei) and the background of the image.

To determine the global threshold, the L_2E algorithm developed by David Scott [59] will be used. Under L_2E, the goal is to minimize the mean integrated square error (MISE) which is given by

$$MISE(\hat{\theta}) = E_{\hat{\theta}} \int \left[f(x|\hat{\theta}) - f(x|\theta_0) \right]^2 dx, \tag{3.51}$$

where $\hat{\theta}$ is the estimate of the parameter vector θ_0 for some parametric probability distribution f, and $E_{\hat{\theta}}$ is the expected value of the integral immediately following. Scott shows the L_2E estimate $\hat{\theta}_{L2E}$ is given by

$$\hat{\theta}_{L2E} = \arg\min_{\theta} \left[\int f(x|\theta)^2 dx - \frac{2}{n} \sum_{i=1}^{n} f(x_i|\theta) \right]. \tag{3.52}$$

A very important property of the L_2E estimator is that in many situations, the L_2E solution finds the largest components [59]. In practice, this means if data is drawn from a well-separated mixture of distributions, the L_2E solution will converge to the original mixture. In other words, the L_2E solution tends to ignore data not drawn from the distribution or mixture defined in (3.52).

It is reasonable to assume the background of the image is drawn from an Exponential distribution with some rate parameter θ. It is also reasonable to assume the nucleus and cytoplasm are each drawn from a Normal distribution. Thus, the

separation property of the L_2E solution can be leveraged to determine not only an estimate of θ, but also a weighting on the data. The weighting can be taken to be the percentage of the pixels which actually fit the Exponential distribution described.

Let ψ be the parameter vector consisting of $[w, \theta]^T$, where w is the weight of the Exponential distribution (between 0 and 1) and θ its rate parameter. It assumed the background of the image follows the density

$$f(x|\psi) = w\theta \exp(-\theta x). \tag{3.53}$$

Now, by (3.52),

$$\begin{aligned} \hat{\psi}_{L_2E} &= \arg\min_{\psi} \left[\int w^2\theta^2 e^{2\theta x} dx - \frac{2w}{n} \sum_{i=1}^{n} \theta e^{-\theta x_i} \right] \\ &= \arg\min_{\psi} \left[-\frac{w^2\theta}{2} - \frac{2w}{n} \sum_{i=1}^{n} \theta e^{-\theta x_i} \right]. \end{aligned} \tag{3.54}$$

As this is a nonlinear optimization problem, it cannot be solved directly. Instead, the Newton-Rhapson algorithm is employed, updating ϕ_t such that

$$\phi_{t+1} = \phi_t - [\nabla^2 g]^{-1} \nabla g, \tag{3.55}$$

where

$$g = -\frac{w^2\theta}{2} - \frac{2w}{n} \sum_{i=1}^{n} \theta e^{-\theta x_i}.$$

It can be shown

$$\nabla g = \begin{pmatrix} -w\theta - \frac{2}{n}\sum_{i=1}^{n}\theta e^{-\theta x_i} \\ -\frac{w^2}{2} - \frac{2w}{n}\sum_{i=1}^{n}(1-\theta x_i)e^{-\theta x_i} \end{pmatrix}, \text{ and} \tag{3.56}$$

$$\nabla^2 g = \begin{pmatrix} -\theta & -2 - \frac{2}{n}\sum_{i=1}^{n}(1-\theta x_i)e^{-\theta x_i} \\ -2 - \frac{2}{n}\sum_{i=1}^{n}(1-\theta x_i)e^{-\theta x_i} & -\frac{2w}{n}\sum_{i=1}^{n}(\theta x_i^2 - 2x_i)e^{-\theta x_i} \end{pmatrix}. \tag{3.57}$$

Using (3.55)-(3.57), (3.54) can be minimized to obtain both the weight $\hat{w}$ and the rate parameter $\hat{\theta}$. In reality, only $\hat{w}$ will be used. Using $\hat{w}$, the number of pixels which belong to the background distribution can be determined. To find the global threshold value, the lowest intensity is found where the number of pixels with that intensity or less is greater than the estimated number of background pixels. Using the threshold obtained from the L_2E procedure, a global thresholding step is performed as described in Chapter 3.3.1.

The cytoplasmic boundaries are determined by employing the procedure in Chapter 3.3.4 to the binary image resulting from the global thresholding step. In the current frame (Figure 3.20), the resulting overlay of tracking regions is given in Figure 3.26.

3.5 Error Metrics

To test the results of the proposed tracking algorithm, a series of nine error metrics were designed. These metrics include the mean integrated percent classification error, as discussed in Chapter 3.5.2, the integrated mean nuclear classification error

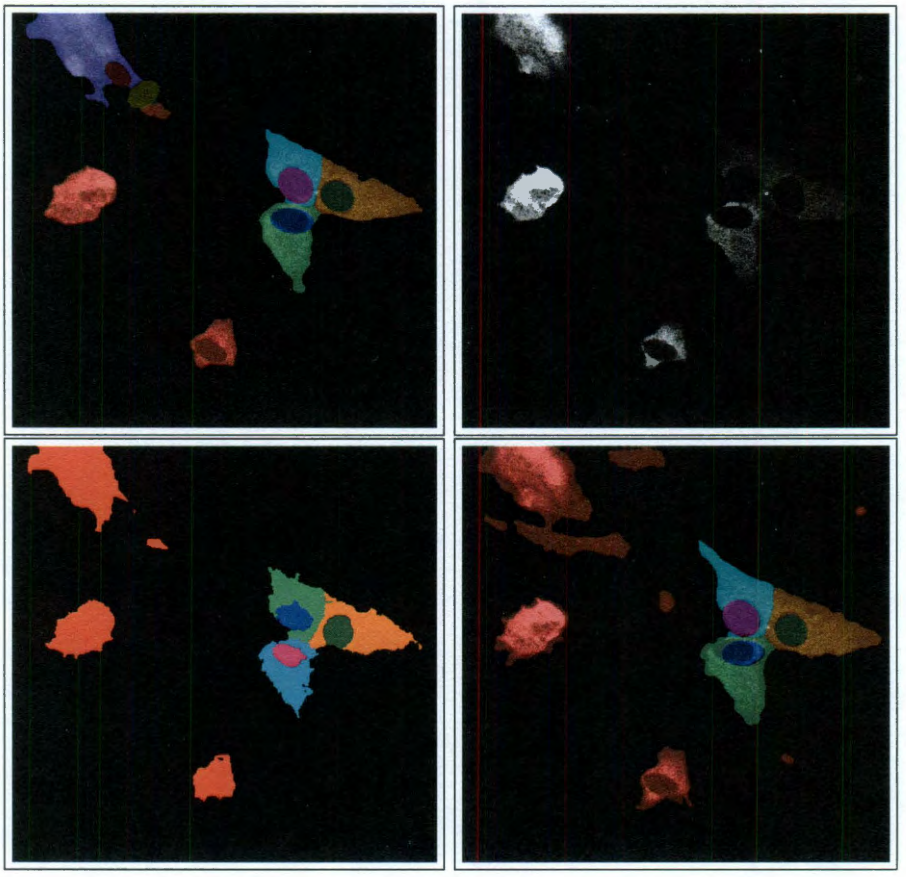


Figure 3.26 : Major points in each iteration of the tracking algorithm. Top left: Final regions for previous frame. Top right: Raw data for current frame. Bottom left: Estimated region assignments for current frame. Bottom right: Final region assignments for current frame.

(Chapter 3.5.3), the integrated mean cytoplasmic classification error (Chapter 3.5.4), and the mean error in the nuclear parameters, which is an amalgamation of several different metrics and is discussed in Chapter 3.5.5. The sources of classification error are summarized in Table 3.1.

However, as the output from the proposed tracking algorithm is a labeled image, the regions in the output had to first be assigned to corresponding regions generated by the simulation. This was done through use of the Hungarian method, and is discussed in Chapter 3.5.1.

3.5.1 Region Assignment

It is common in the simulations to have cells far enough outside the image boundaries for the tracking algorithm to be unable to detect them. Similarly, although the tracking algorithm uses integer labels for each region - an integer label that corresponds to an RGB value - the output from the simulations is a raw RGB value. Therefore, the first step in calculating error metrics is to find a method by which RGB values from the tracking algorithm can be mapped to RGB values from the simulation.

The Hungarian method was used to associate these regions. The Hungarian method was developed by Harold Kuhn [37] to solve the assignment problem. The Hungarian method accepts an $n \times n$ matrix and returns the assignment of n rows to n columns with the minimum cost. The implementation used is a negligible modification of the implementation done by John Weaver [71], which is posted online.

To calculate error, every non-black RGB value in the output of the tracking algorithm is assumed to correspond to a non-black RGB value in the simulation. However, due to the concerns mentioned above, the reverse cannot be assumed. Therefore, if there are t distinct regions T_i in the tracking results (including the background) and s distinct regions S_i in the simulation (including the background), then the cost matrix $\boldsymbol{C}$ for the Hungarian method is of size $\max(t, s) \times \max(t, s)$. The entries of the $\boldsymbol{C}$ are defined as

$$\boldsymbol{C}_{i,j} = \left|[T_i] - [S_j]\right| + ||\bar{T}_i - \bar{S}_j||,$$

where $[X]$ denotes the number of pixels in region X, and $\bar{X}$ denotes the calculated center of region X. At the same time, an overlap matrix $\boldsymbol{O}$ of the tracked regions T_i is generated so that if $AABB_i$ is the axis-aligned bounding box of region T_i, then

$$\boldsymbol{O}_{i,j} = [AABB_i \cap AABB_j].$$

After running the Hungarian method against the cost matrix C, the returned assignment matrix C' is a matrix where every entry is either 0 or -1. The tracked regions T_i are mapped to the simulation regions S_j by finding the zero coefficient for each row i in the matrix resulting from the Hungarian method. The column index j of the zero coefficient indicates which region S_j T_i is mapped to.

For each tracked region T_i, the region T_j is found which has the greatest overlap with T_i using the overlap matrix $\boldsymbol{O}$. The smaller of T_i and T_j is taken to be the nucleus, and the larger to be the cytoplasm of the cell in the tracking results. The corresponding S_k and S_l are drawn from the previously described mapping as the

simulation cytoplasm and nucleus.

This method is run only for the first frame in the time series. The remaining frames leverage these results to determine which RGB values apply to the different regions in each frame.

3.5.2 Mean Integrated Percent Classification Error

The mean integrated percent classification error (MIPCE) yields a measure of how well the tracking algorithm performs overall. It is calculated as follows:

For every frame, every pixel is examined. If the assigned region T_i from the tracking result image does not match the mapped region S_j for the same pixel in the simulation image, it is considered to be an error. The integrated classification error up to time t is given by

$$ICE_t = \sum_{i=1}^{t} E_i,$$

where E_i is the number of errors in a given frame. The final mean integrated percent classification error (MIPCE) is the obvious

$$MIPCE = \frac{1}{T|I|} \sum_{i=1}^{T} E_i, \tag{3.58}$$

where $|I|$ is the number of pixels in each frame.

3.5.3 Integrated Mean Nuclear Classification Error

The integrated nuclear classification error (IMNCE) is a fast measure of how well the proposed algorithm tracked the nuclear boundaries in a given frame. There are several

errors which are considered to be a nuclear classification error, which are summarized in Table 3.1. The first is if the tracking region R_i indicates a nuclear pixel, but the simulation region S_j indicates any other region. Similarly, if the simulation region S_j indicates the nucleus defined by R_i, but the tracking region R_k does not, then the error is considered to ba nuclear error. Thus, for any frame t, in a series where n cells were tracked,

$$MNCE(t) = \frac{1}{n}\sum_{i=1}^{n} E_{N_i},$$

where E_{N_i} is the number of errors relating to the nuclear region of cell i. Once again, the obvious resulting error metric IMNCE is given by

$$IMNCE(t) = \sum_{t=1}^{T} MNCE(t). \tag{3.59}$$

Because these are classification errors, the error metric is a monotonically non-decreasing function. The monotonicity allows for the location of sudden spikes in the IMNCE, which are indicative of a general mass failure in the tracking algorithm. In addition, a total error over the course of the run can be calculated for fast comparison. The use of the mean error between nuclei in an image has been chosen because it is not out of the realm of possibility that different tracking runs of the same data may not pick up the same number of cells due to the stochastic nature of the initial segmentation. This allows for the comparison of these multiple runs to obtain an overall error metric for tracking the simulation.

3.5.4 Integrated Mean Cytoplasmic Classification Error

The integrated mean cytoplasmic classification error (IMCCE) is very similar to the IMNCE, except that it measures only the accuracy of the cytoplasm-background and cytoplasm-cytoplasm boundaries. The calculation of the IMCCE is equivalent to the calculation of the IMNCE with one exception. If the tracking region R_i indicates the cytoplasm, and the simulation region S_j indicates the nucleus of the cell associated with R_i or vice versa, then the error is disregarded for this metric. This is done because the goal of this metric is to determine the error in the cytoplasmic boundary as opposed to the nuclear boundary. Once again, the sources of error are summarized in Table 3.1

		Assignment from Simulation Image				
		Background	Nuc (Cell I)	Cyt (Cell I)	Nuc (Cell J)	Cyt (Cell J)
Assignment from Tracking Image	Background	NO ERROR	Total, Nuc I	Total, Cyt I	Total, Nuc J	Total, Cyt J
	Nuc (Cell I)		NO ERROR	Total, Nuc I	Total, Nuc I, Nuc J	Total, Nuc I, Cyt J
	Cyt (Cell I)			NO ERROR	Total, Cyt I, Nuc J	Total, Cyt I, Cyt J

Table 3.1 : Error sources for classification error. Cyt is short for cytoplasm, Nuc is short for nucleus. The error sources are symmetric, so the lower triangular region is left black for clarity.

3.5.5 Mean Nuclear Parameter Error

The mean nuclear parameter error is a direct measure of how well the particle filter algorithm from Chapter 3.4.6 locates the true parameters of the nucleus. Although the mean nuclear parameter error ties directly in with the IMNCE, the performance of the particle filter algorithm itself must be taken into account. Therefore, six error measurements based on ellipses fit to the boundaries of corresponding regions are tested. That is, if region T_i corresponds to a nuclear region defined by S_j, then ellipses e_S and e_T will be fit to the boundaries of S_j and T_i, respectively. The ellipse fitting follows the procedure outline in Chapter 3.3.3.

The first two error metrics are the signed error in the x and y direction between the center of the nucleus as calculated in the tracking algorithm and the true center of the nucleus from the simulation. By averaging over these signed errors, it is possible to determine any bias within the tracking algorithm itself.

The next two measurements are the percentage deviation in the estimates of the major and minor axis half length. The absolute deviation is calculated as taking the absolute difference between the half-lengths of the axes fit to the tracking results and the half-lengths of the axes fit to the simulation data. The deviation is divided by the half-length of the appropriate axis fit to the simulation data to obtain the percent deviation. Once again, the goal is to detect bias and overall trends.

The fifth measurement is the signed error in the estimation of the ellipse rotation angle. This is not as simple as the signed error in the x and y coordinate due to the

symmetry of ellipses around their major and minor axes. As an example, it is easy to see that rotations of θ and $\pi + \theta$ radians yield the same ellipse. Because of symmetry, the maximum rotation error occurs at $\pm\pi/2$ radians, or ± 90 degrees. Therefore, the error for the rotation θ is calculated as

$$E_\theta = \begin{cases} -\pi - (\theta_T - \theta_S) & : \theta_T - \theta_S < -\frac{\pi}{2}, \\ \theta_T - \theta_S & : -\frac{\pi}{2} \leq \theta_T - \theta_S \leq \frac{\pi}{2}, \\ \pi - (\theta_T - \theta_S) & : \theta_T - \theta_S > \frac{\pi}{2}, \end{cases}$$

where θ_T is the rotation angle of the fitted ellipse e_T and θ_S is the rotation angle of the fitted ellipse e_S, both of which are in radians.

The final measurement is the distance between the center of the ellipse e_T and the center of the ellipse e_S as a percentage of the length of the radius on e_S on which e_T lies, which is depicted more clearly in Figure 3.27. This metric was designed to yield a better picture of how close the proposed algorithm is to the true value than the signed distance in the x and y directions. By representing the error as a percentage of the elliptical radius, the error in the nuclear localization tends to be overestimated. However, it also allows fast recognition of w when a tracking result is sufficiently far enough off so that the center of the e_T is not even contained within e_S.

The simplest way to calculate this final measurement is to first translate both ellipses as expressed in Figure 3.27 so that the center of e_S is located at the origin. The ellipse e_T is rotated around the origin by $-\theta_S$. If the ellipse e_S is presented in

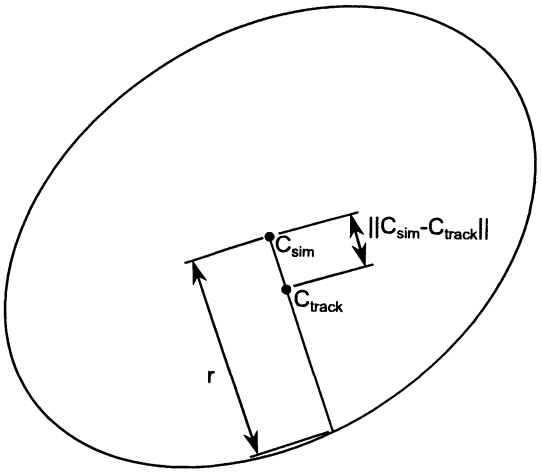


Figure 3.27 : Ellipse from simulation shown. C_{sim} is the calculated center of the ellipse e_S from the simulation. C_{track} is the calculated center of the ellipse e_T (boundary not shown) from the tracking algorithm. The distance metric is given by $||C_{sim}-C_{track}||/r$.

polar form, then

$$r(\theta) = \frac{ab}{\sqrt{(a\sin\theta)^2 + (b\cos\theta)^2}}, \tag{3.60}$$

where a and b are the half-lengths of the ellipse e_S. However, for the radius which passes through the translated and rotated center of e_T (denoted C_t'),

$$\theta = \arctan\frac{C_t'(y)}{C_t'(x)}. \tag{3.61}$$

Now, (3.60) and (3.61) can be used to calculate the percent distance error measurement.

These nuclear localization errors will be averaged over all of the cells in a single frame due to the variable number of cells which may be detected in different runs of

the same data. However, the nuclear localization errors will not be integrated over the entire time series.

Chapter 4

Simulation of 2D Cell Movement

The movement of cells across a surface is a highly complex mechanism which includes several distinct stages [48, 73]. The first stage consists of the protrusion of a portion of the cell membrane (a mechanism driven by the formation of actin and myosin networks [48]). The second stage of cell crawling is the adhesion of the protrusion - the lamellipodium - to the surface substrate. After adhesion has occurred, the adhesive bonds holding the cell body and the rear of the cell are released. The actin and myosin filaments in the lamellipodium contract, bringing the remainder of the cell body into the space occupied by the lamellipodium.

Several different groups have simulated the movement of cells across a surface, including Mogilner et. al, [47, 48], Zhou [73], and Schreiber et. al [58]. Mogilner focused on the physics of the lamellipodiar protrusion, preferring to use a finite element simulation to calculate the force generating the lamellipodium [48]. Mogilner uses these calculated forces to generate a one-dimensional simulation of a crawling cell [47] based on the biophysical model laid out by Bottino et. al [7]. Zhou expands this model into a moving-boundary problem in an effort to move from a discretized to a continuous model [73]. In both works by both Mogilner and Zhou, a constantly changing unstructured mesh is used as the underpinnings of the finite elements sim-

ulation. For Zhou, this is done by generating a two-dimensional domain in which the actual boundary will lie and using finite elements to determine the exact path of the boundary. For Mogilner, an adaptive mesh is used for the lamellipodium, in which nodes are added and subtracted as needed to correspond to the creation and contraction of the actin and myosin filaments.

Schreiber et. al built a three-dimensional simulation of the lamellipodium and subsequent cell movement based on the physical structure of F-actin [58]. In their model, the growth, capping and decomposition of the branched F-actin polymers are modeled inside a box with edges being the top, bottom, and cytoplasmic wall of the cell.

Other simulations of cell movement tend to focus on the path taken by the cell and cellular interactions, such as Taylor et. al [63] and Umeda and Inouye [65]. Umeda chooses to simulate cells as rigid particles which interact, as they are more interested in the result of cellular interaction. Taylor, et. al was also interested in the path of cell movement, as opposed to the mechanics thereof. However, Taylor et. al chose to implement their cells as a physical soft body. A ring of point masses with nonzero radii connected to neighboring masses with springs was used to model the cell body. Movement was simulated by allowing each point mass to vary in position. Taylor et al also studied the effects of cell-cell adhesion by forming temporary bonds between point masses of different cells. [63].

However, all of these simulations, with the exception of Taylor et al, and Umeda

and Inouye are built to simulate the movement of cells at with high accuracy. In addition, Mogilner and Zhou's simulation methods account for only a single cell, which naturally will not cover the resolution of cell-cell interactions and collision. One method which could be used to bypass this problem would be to create a mesh out of the entire simulation world, as was done in Cardoze et. al [12]. Cardoze et. al's work simulates the movement of multiple cells by using a separate triangular mesh for each cell. These meshes will be deformed as necessary to obtain realistic cellular movement.

The goal of the simulation presented here is speed and trackable nuclear and cytoplasmic boundaries as opposed to an exact model of cellular movement. To accomplish this goal, the simulation program takes a step back from Zhou and Mogilner, completely discretizing both the structure of the cell and the time sequence of cell movement. While it is understood that *in vivo*, the different stages of cell movement happen nearly simultaneously, the protrusion, movement of cell body, and movement of the rear of the cell will be separated into three distinct stages. The simulation will also differ from Mogilner by representing each cell as a static unstructured triangular mesh for each movement cycle. The boundary of the nucleus and cytoplasm will be retained and the interior of the cell will be remeshed between each movement cycle for computational and physical reasons.

The creation of the initial boundaries is described in Chapter 4.1. The creation, refinement, and additions to the driving mesh are examined in Chapter 4.2. Chapter

4.3 discusses the theory behind the movement of the cell and how the different phases of cell movement will be simulated, and Chapter 4.4 describes the mathematical details of the simulation. Chapter 4.5 describes how the simulated data sets are obtained from the mesh at any given time step.

For this simulation, the Mersenne Twister random number generation algorithm [45], as implemented by Jasper Bedaux [5], is used to add stochasticity to the simulation. The Mersenne Twister algorithm provides a random number generation with a high period. As random values will be generated for each pixel in the time series as well as spring length multipliers for a large number of springs, a long period is essential for a high quality simulation.

4.1 Generation of Random Boundaries

To obtain a reasonable facsimile of two-dimensional cell movement across a surface, the first step is to obtain an amorphic cell-like boundary to simulate the cell membrane and an enclosed ellipse to simulate the cell nucleus. The simulation of a cytoplasmic boundary is a three-step process described in Chapter 4.1.1. The simulation of the nuclear boundary is described in Chapter 4.1.2

4.1.1 Generation of Amorphic Cytoplasmic Boundary

The amorphic cell boundary is based on a regular polygon. For the illustrated example, a regular hexagon was used. The center of the polygon was selected as a uniform

random point in the allowed image field, which for the simulations described, ran from $(0,0)$ to $(511,511)$. For each corner, a random polar value (r,θ) is generated. The radius value r is a uniformly distributed floating-point value which determines the distance from the center of the polygon to the corner point. The angle value θ is generated from a normal distribution defined such that the mean is the "standard" angle for a normal polygon, but the variance is defined so that the corners remain in "order," i.e. $(\theta_0 < \theta_1 < \ldots < \theta_{n-1})$. More concretely, if a regular hexagon is used as the base polygon, the angle between corner points is 60°. Without loss of generality, define $\theta_0 = 0°$, $\theta_1 = 60°$, etc. for a regular hexagon. To retain this order, the standard deviation σ for all of the corner points is set to be 10°. Thus, 99.7% of the time the calculated polar angle will fall the ranges for a random hexagon listed in Table 4.1. A graphical view of the probability ranges is included as Figure 4.1a. In this figure, the random center of the hexagon is the large dot in the middle. Each solid line is the mean of the defined normal distribution with the dotted lines begin the expected edges of the distribution at 3σ. The brightness of the color is indicative of the likelihood of that point being generated. The three small black dots are the three randomly generated corner points $\boldsymbol{A}, \boldsymbol{B}$, and $\boldsymbol{C}$.

The second step of the cell boundary generation process is to select control points, as shown in Figure 4.1b. To generate these control points, each pair $(r,\theta)_i$ and $(r,\theta)_{i+1}$, with $(r,\theta)_n = (r,\theta)_0$ is used to define a rectangular support for a two-dimensional uniform distribution in Cartesian coordinates. Two ordered points are

Corner	Mean	Std. Dev	Low (°)	High (°)
0	0	10	330	30
1	60	10	30	90
2	120	10	90	150
3	180	10	150	210
4	240	10	210	270
5	300	10	270	330

Table 4.1 : Corner, Mean, Standard deviation, and corresponding high and low values for θ_i for an amorphic cell based on a regular hexagon.

generated from each uniform distribution defined by $[(x,y)_i, (x,y)_{i+1}]$. Figure 4.1b shows the generated points, labeled **1, 2, 3** and **4** generated by the corner pairs $(\boldsymbol{A}, \boldsymbol{B})$ and $(\boldsymbol{B}, \boldsymbol{C})$.

The third step of the boundary generation process is to generate a Bézier curve from each of the quartets of points consisting of $[(x,y)_i, (x,y)_{CP1}, (x,y)_{CP2}, (x,y)_{i+1}]$, where $(x,y)_{CP1}$ and $(x,y)_{CP2}$ are the control points determined in step 2. Each of the ordered quartet of points can be used to draw a Bézier curve using the parametric equation

$$\mathcal{B}(x,y) = (1-t)^3(x,y)_i + 3t(1-t)^2(x,y)_{CP1} + 3t^2(1-t)(x,y)_{CP2} + t^3(x,y)_{i+1},\ t \in [0,1] \tag{4.1}$$

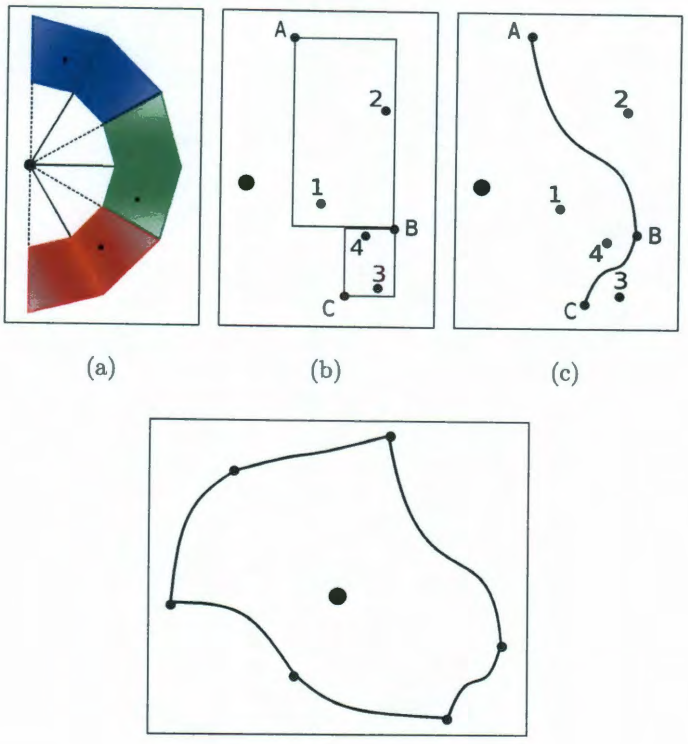


Figure 4.1 : A visual description of the process of generating an amorphic cell boundary. The large point is the random center of the cell. **(a):** The brightness of each region corresponds to the likelihood of selecting a corner point at that location. The solid lines are the "regular polygon" corner values, with the dashed lines being the limits on where the values can be chosen. The small black dots are the randomly selected points. **(b):** Two random points are chosen in the rectangle defined by neighboring corner points from (a). **(c):** The Bézier curves (cubic splines) drawn using the control points from (b). **(d):** The resulting amorphic cytoplasmic outline.

The curves arising from corner points $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ and control points **1**, **2**, **3**, and **4** are shown in Figure 4.1c.

The Bézier curve generation process is followed for all the regular polygon corner points. The Bézier curves generated here represent the initial state of an amorphic cell boundary, as shown in Figure 4.1d. The cytoplasmic boundary is rotated by a random uniform angle to complete the boundary generation.

4.1.2 Generation of Nuclear Boundaries

To simulate nuclei, a random ellipse needs to be generated to fit within the amorphic cellular boundary from in Chapter 4.1.1. One of the primary inputs for the generation of the nuclear boundaries is the axis-aligned bounding box of the cellular boundary, which is defined as the rectangle made up of the minimum and maximum X values (width) and the minimum and maximum Y values (height), denoted $AABB$. In addition, the number of points enclosed by the cellular boundary, denoted n, is also calculated.

Recall from Chapter 3.3.3 that any ellipse can be uniquely represented by five parameters. To build a random ellipse, a random X value is generated from a normal distribution with μ_x given by the "center" of the cell chosen in Chapter 4.1.1 and standard deviation as one tenth the width of $AABB$. This value is set as the x value of the nuclear origin. Similarly, the ellipse center y value is taken from a normal distribution with μ_y given by the "center" of the cell and standard deviation as one

tenth the height of $AABB$.

The nucleus is assumed to cover between 17 and 25 percent of the surface area of a crawling cell. The major axis of the ellipse representing the nucleus is also assumed to be no less than one-fourth the larger of the width of $AABB$ or the height of $AABB$. Knowing these two pieces of information, the possible range for the major axis half-length is defined as the following:

$$\text{Minimum Nuclear Area} = \frac{n}{6}$$

$$\text{Maximum Nuclear Area} = \frac{n}{4}$$

$$\text{Nuclear Area} = \pi * R_x * R_y$$

$$\text{Minimum Rad} = \frac{1}{4}\max\left[\text{width}(AABB), \text{height}(AABB)\right]$$

$$\text{Maximum } R_x = \frac{\text{Maximum Nuclear Area}}{\pi * \text{Minimum Rad}}$$

$$\text{Minimum } R_x = \sqrt{\frac{\text{Minimum Nuclear Area}}{\pi}}$$

A value for R_x is selected from a uniform distribution between "Minimum R_x" and "Maximum R_x." A minimum and maximum value for R_y is defined as:

$$\text{Maximum } R_y = \frac{\text{Maximum Nuclear Area}}{\pi * R_x}$$

$$\text{Minimum } R_y = \frac{\text{Minimum Nuclear Area}}{\pi * R_y}$$

R_y is chosen from a uniform distribution between "Minimum R_y" and "Maximum R_y." The final step in the potential nuclear generation process is the rotation of the nucleus by a random angle between 0 and 360 degrees.

After selecting these parameters, a series of four Bézier curves is generated to approximate the outline of the ellipse. If any of these curves intersect the Bézier curves defining the cellular boundary a new random ellipse is generated. If ten attempts to generate nuclear boundaries fail for a given cellular boundary, the cellular boundary is discarded and a new one created.

4.2 Mesh Generation

To simulate cell movement across a two-dimensional surface, the cell is represented as a dual-layer Hookian mass-spring network. The primary layer is built from a Delaunay Triangulation of the cellular and nuclear boundaries of the cell. The secondary layer is a series of springs connected the center of the nucleus. The formation of the primary layer is described in Chapters 4.2.1 and 4.2.2. The formation of the secondary layer is described in Chapter 4.2.4. In Chapter 4.2.5, the method used to recreate the mesh structure and the reasoning for recreation are described. The actual mechanics of the system are described in Chapter 4.3.

A Delaunay Triangulation is a method of triangulating a set of points where no point P is located within the circumscribed circle of any triangle T in the triangulation of which P is not a vertex. The Delaunay Triangulation was first proposed by Boris Delaunay [18] in 1934. By constraining the interior angles of the triangulation and adding additional points as necessary it is possible to obtain a high-quality triangular mesh from points along the outline of an object.

4.2.1 Initial Mesh

The initial mesh is formed in two steps. First, the cellular and nuclear boundaries are linearized as described in Chapter 3.3.3 with a very small error threshold. The Triangle library contains algorithms for generating high quality triangular meshes [62]. A very high quality mesh is preferred for the initial generation, so constraints are placed on the triangulation to prevent triangles with interior angles of less than 30 degrees. The Triangle library fulfills the quality constraint by continually subdividing triangles until the constraint holds. However, due to the limitations of floating-point precision, it is not always possible to satisfy this constraint. To compensate for situations where a high quality mesh cannot be generated, attempts are made to limit to triangles with no interior angles of less than 25 degrees, then by 20 degrees if necessary. If both of these options fail, it is assumed the mesh is impossible to build, and an entirely new cell is built. The results from the Triangle algorithm are used to build a cellular mesh structure, which is implemented using the OpenMesh library [6].

Figure 4.2 shows an example of a full randomly generated amorphic cellular boundary with an elliptical nucleus. This boundary is linearized and transformed into a triangular mesh as described above. The resulting mesh is featured in Figure 4.3. However, the initial mesh as shown has a number of high density vertex clusters. The presence of these clusters causes two problems with the simulation. First, a high concentration of mesh vertices in a single area causes artificially low fluidity in that

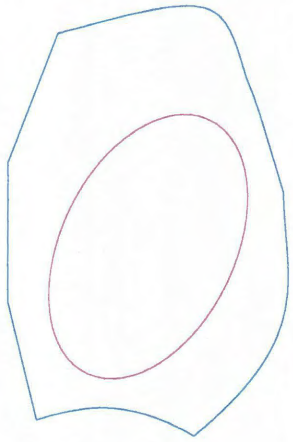


Figure 4.2 : Fully amorphic cellular boundary based on hexagonal base.

portion of the cell membrane, due to the high number of springs and the corresponding low number feasible areas in which each vertex can be placed for each iteration. Second, a high concentration of springs increases the possibility of "tangling" the mesh, which will be discussed in Chapter 4.4.3.

4.2.2 Mesh Decimation

To combat the problem of high vertex concentration, a process of mesh decimation is employed. The algorithm for mesh decimation was entirely implemented in the OpenMesh library [6]. The decimation algorithm is a greedy algorithm, where each edge is scored based on a specific error metric, which is an indication of how important the edge is to the overall structure of the mesh. As long as certain conditions are not met, the algorithm collapses the edge with the highest error. When an edge is collapsed, the two vertices that make up the edge are joined into a single vertex at

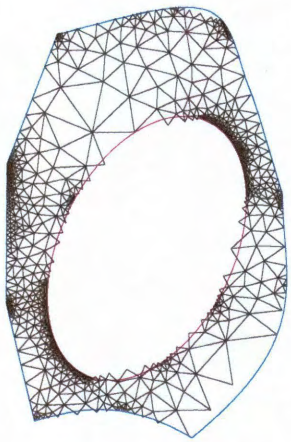


Figure 4.3 : Initial triangular mesh. The high volume of nodes in the areas of high curvature generate an artificially high stiffness in the mesh. The mesh must be thinned in a reasonable manner to alleviate the stiffness.

their midpoint, and the remaining edges adjust accordingly. The error metric used is a quadric error method designed by Michael Garland [23], which allows for a fast method of calculating the squared distance from a point to multiple lines. The total quadric error becomes the sum of these squared distances.

In the simulation, the user is allowed to specify both a maximum quadric error and a minimum number of nodes. The simulation program will decimate the mesh up to the maximum quadric error, unless removing a node would result in fewer than the specified minimum number of nodes. If the minimum number of nodes is reached, the simulation would stop removing nodes. As an example, to remove the high node density clusters in Figure 4.3, the maximum quadric error is set to 500

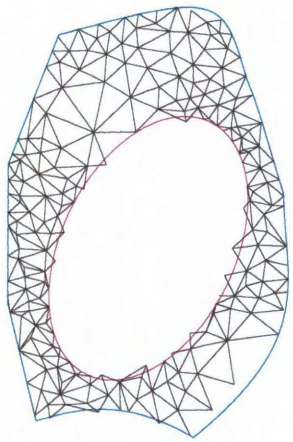


Figure 4.4 : Mesh after quadric decimation (maximum error 500). Note the thinning of the mesh without the loss of boundary detail.

with a minimum of 100 nodes. The resulting decimated mesh is shown as Figure 4.4.

4.2.3 Nucleus

In most cases the procedure described in Chapters 4.2.1 and 4.2.2 is sufficient to simulate the random changes in the cytoplasmic boundary of a cell. However, by varying the lengths of the mesh edges (springs) as is planned, the vertices surrounding the nucleus have the ability to encroach into the nuclear area. However, the nucleus in a viable cell is elliptically shaped, so random encroachments can not be tolerated in the simulation of the nuclear membrane. To prevent the loss of the nuclear elliptical shape, the nucleus is constrained to be a rigid elliptical body. The details of this constraint are covered in Chapter 4.4.3.

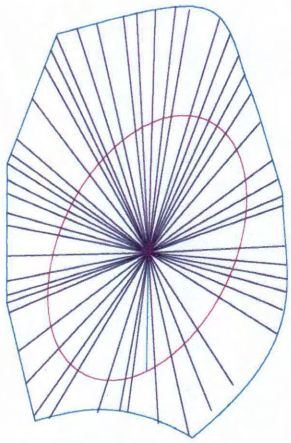


Figure 4.5 : Second spring layer for the movement simulation. Each spring stretches from the center of the nucleus to a cytoplasmic boundary point.

4.2.4 Nuclear Springs

Early trials of the simulation showed a possibility of unreasonable concavity in the simulated cellular membrane due to the method of modeling the extrusion of lamellipodia. This limitation was overcome by imposing a second set of springs on the system. These springs originate from a common point at the center of the nucleus and radiate outwards to each point on the cytoplasmic boundary. The spring constant for each of these is set to one tenth of the user entered default spring constant. The lower spring constant for this layer of springs allows for reasonably slow movement of the nucleus through the simulated cell structure. An example of the nuclear spring system is shown in Figure 4.5.

4.2.5 Remeshing

After each cell movement cycle, i.e. once the cell has returned to a resting state, the entirety of the cell is remeshed. This remeshing is accomplished by approximating the current cellular boundary using a series of Bèzier curves to smoothly interpolate between the boundary mesh points. Similarly, the nucleus is refit as a parametric ellipse using the algorithm from Chapter 3.3.3. A new initial mesh is created using these boundaries, and decimated as before.

The remeshing procedure prevents the cell from resuming its initial shape during the rest period between movement cycles. The remeshing also keeps the number of vertices in the mesh to a sufficiently high value to retain smoothness in the simulated cellular boundary.

4.3 Cell Movement Mechanics

As described in Chapter 4.2, each cell is represented by a double-layer, ideal, undamped mass-spring system. In Chapter 4.3.1, the physics of an ideal Hookian spring and the definition of masses and springs in the mesh will be given.Chapter 4.3.2 describes how a lamellipodium is generated within the mesh. Chapters 4.3.3 through 4.3.5 describe the different phases of a mobile cell crawling across a two-dimensional surface. Chapter 4.3.6 describes how cells are modeled to have random fluctuations in the cell membrane without actually undergoing defined movement.

4.3.1 Hookian Springs

In Newtonian physics, a one-dimensional Hookian spring is a linear object where the forces on the two ends are directly proportional to the strain placed on the object, i.e.

$$F = k|l - l_0|, \tag{4.2}$$

where F is the magnitude of the force acting on each end of the object, l is the current length of the object, and l_0 is the resting length of the object. The direction of the force for each end of the object is set so the points always move back toward the resting length. For a two-dimensional spring, the force is given by

$$F_i = k\frac{\boldsymbol{x}_j - \boldsymbol{x}_i}{||\boldsymbol{x}_j - \boldsymbol{x}_i||}(||\boldsymbol{x}_j - \boldsymbol{x}_i|| - l_0), \quad F_j = -F_i \tag{4.3}$$

with $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ being the the two endpoints of the spring. The difference between (4.2) and (4.3) form is the presence of the direction term, given by the unit vector in the direction of $\boldsymbol{x}_j - \boldsymbol{x}_i$. Therefore, if the distance from $\boldsymbol{x}_i$ to $\boldsymbol{x}_j$ is greater than l_0, the force on $\boldsymbol{x}_i$ moves it toward $\boldsymbol{x}_j$. At the same time, the direction of the force of $\boldsymbol{x}_j$ is in the direction of $\boldsymbol{x}_i$.

In the simulation, the edges of the mesh are considered to be idealized Hookian springs. The length of the edge is defined to be l_0, the resting length for each underlying spring. The spring constant is also modified to allow for variable stiffness within the cell. Namely, smaller springs are manipulated for increased stiffness, which helps avoid tangling the mesh and losing the cohesiveness of the system. The spring

constant manipulation consists of dividing the user-supplied base spring constant by l_0 for each spring. For simplicity, a uniform mass over the entire cell is assumed. Although the assumption is not entirely valid, the results show a visually reasonable approximation to cell crawling mechanics.

4.3.2 Lamellipodium Generation

Lamellipodia are the flat forward extensions of crawling cells. The lamellipodium machinery is formed of a mixture of actin and myosin proteins along a radial track through the lamellipodium itself [48, 47], while are generated by the polymerization of actin proteins at the leading edge of the lamellipodium.

The lamellipodia is generated by determining a random direction and angle. The base direction is sampled uniformly in integer degrees, which is randomly changed up to a variance given by the user. The random change in the base direction allows for the creation of multiple non-overlapping lamellipodia during a single movement phase. A width in degrees is randomly chosen from a normal distribution defined by the user for each lamellipodia. Because the base direction and width of the lamellipodium is known, a triangular region can be generated to encompass all of the "cytoplasmic mass" contained within the given lamellipodium. For every node contained within this triangular region, it is assumed it and any other nodes connected directly with it are part of the lamellipodium, so long as those nodes are not on the nuclear boundary.

Knowing the directional limits of the forward movement, a triangular region can

be generated in the opposite direction encompassing the same angular width, which is considered to be the "rear" of the cell. If a node is contained within the second triangular region, it is considered to be a "rearguard" point, as long as it is not on the nuclear boundary.

4.3.3 Lamellipodium Extension

The extension of the lamellipodium by the growth of actin tracks is simulated by extending any springs which connect to pointmasses assigned to the lamellipodium. That is, for each edge in the mesh which has one node contained in the forward triangular region defined in Chapter 4.3.2, the resting length l_0 of the edge is multiplied by a random number. This number is generated from a uniform distribution with the minimum and maximum values supplied by the user, and is the random number is independently sampled for each edge. Similarly, for each boundary vertex contained within the lamellipodium, the resting length of the nuclear spring associated with that vertex is also increased by an independent random number from the distribution just described.

It is known that during lamellipodiar extension, the main body of the cell, the nucleus and organelles, does not move. To prevent the extension of the springs pushing back against the nuclear location, vertices which make up the nuclear boundary or are considered "rearguard" points are assigned to be "stuck." More information on "stuck" points is given in Chapter 4.4.1.

4.3.4 Nuclear Movement

In a cell, the movement of the primary cell mass is due to the contraction of the actin filaments in the lamellipodium [48], which is simulated in two stages. First, the boundary nodes in the lamellipodium are set to to be "stuck", and the nodes defining the nuclear boundary to be "unstuck". The status change allows the nuclear boundary points to move while leaving the boundary of the cytoplasm at its full extension. All the springs in the lamellipodium are contracted by decreasing their initial length by a set amount (usually to half the initial length), where the initial length is the length from the initial mesh, as opposed to the rest length from the extension.

Concurrently, all the springs contained in the area opposite the lamellipodium are extended. The extension has been added only for visual accuracy. The method of extension for the springs in the rear of the cell follows the method described in Chapter 4.3.3.

4.3.5 Rear Boundary Contraction

The final portion of the cell movement phase is the contraction of the rear cytoplasmic boundary. The contraction of the rear cytoplasmic boundary is accomplished by first setting all the nuclear boundary nodes to be "stuck", and setting all the boundary nodes in the "rearguard" area to be "unstuck". The resting lengths for springs in the rear area are set as some predefined fraction of the initial lengths. The forces

generated from the springs in the rear area will cause the contraction of the rear boundary.

4.3.6 Random Cell Boundary Deformation

In vitro studies of cellular movement indicate a lack of consistency in the shape of cytoplasmic boundaries. The simulation models this through a type of cell deformation which is termed "jitter." The jitter deformation is nothing more than a temporary random cell boundary deformation. This jitter deformation is used as an approximation of random growth and degradation of portions of the cytoskeleton when the cell is not in a motile phase.

The jitter phase is performed by choosing an independent random number from a uniform distribution defined by the user for each edge in the mesh. Each edge is either multipled or divided by the uniform number based on the results of an independent Bernoulli random variate for the current edge. The nuclear boundary, being modeled as a rigid structure, is exempt from the jitter movement stage.

4.4 Timestep Updates

This chapter discusses the mathematics behind the simulation. Chapter 4.4.1 describes the variation on Hooke's law used to change the mesh structure. Chapter 4.4.2 gives the mathematical details and reasoning behind the choice of numerical integration schemes and how exactly the implementation is enforced. Chapter 4.4.3

describes the implementation of certain constraints to enforce the relative accuracy and enhance the stability of the simulation.

4.4.1 Force Calculation

There is only one source of force acting on the interior nodes of each cell: the force due to the springs connected with the given node. Nodes on the nuclear and cytoplasmic boundaries, however, have multiple types of forces acting upon them.

Spring Forces

The primary source of force for most nodes is due to the compression and extension of the springs defined by the edges connecting the nodes. For each edge, as discussed in Chapter 4.3.1, the magnitude and direction of the force exerted on each of its nodes is given by (4.3). The data structure used for the mesh is designed with oriented edges in mind, so each edge has a specific "from" and "to" vertex, say A and B. Thus, for any edge $\overline{AB}$, if $\boldsymbol{x}_A$ is the position of point A and $\boldsymbol{x}_B$ is the position of point B,

$$F^s_{A,B} = k\frac{\boldsymbol{x}_B - \boldsymbol{x}_A}{||\boldsymbol{x}_B - \boldsymbol{x}_A||}(||\boldsymbol{x}_B - \boldsymbol{x}_A|| - l_0), \quad F^s_{B,A} = -F^s_{A,B}, \tag{4.4}$$

where $F^s_{A,B}$ is the force acting on A due to the compression or extension of $\overline{AB}$ and $F^s_{B,A}$ is the force acting on B due to the same.

Pressure Forces

It is assumed that a moving cell will keep an approximately constant surface area. A pressure force applied to the boundary nodes is used to fulfill this assumption. The pressure force is derived as in Matyka and Ollila [46].

According to the Ideal Gas law, the pressure P of a given number of molecules of the gas n contained within a certain volume V at a certain temperature T can be calculated in three dimensions as

$$PV = nRT. \tag{4.5}$$

The Ideal Gas Law is assumed to hold for each cell. In addition, the number of "gas" molecules and the temperature are assumed stay constant over time. These assumptions indicate the pressure can be calculated as

$$P = \frac{k_P}{V}$$

for some k_P. For the two-dimensional case, the same relationship applies, replacing volume with area [46].

In the three-dimensional case, the definition of pressure is force per unit area, which corresponds directly to the two-dimensional case of pressure being defined as force per unit length. Hence the magnitude of the force acting on the two endpoints of an edge of length l is given by

$$F^p = lP = l\frac{k_P}{A}, \tag{4.6}$$

where A is the area enclosed by the entire boundary. It is assumed there is no pressure-based force acting on the cytoplasmic nodes when the cell is in its initial configuration. Therefore, it is possible to represent the final pressure-based force magnitude as

$$F^p_{A,B} = (||\boldsymbol{x}_B - \boldsymbol{x}_A||k_P)\left(\frac{1}{\boldsymbol{A}} - \frac{1}{\boldsymbol{A}_0}\right), \tag{4.7}$$

where k_P is predefined by the user, with $\boldsymbol{A}$ being the area currently enclosed by the cell, and $\boldsymbol{A}_0$ as the area enclosed by the cell with the entire spring network at resting lengths. The direction of the pressure force is defined to be outward along the normal to the edge defined from edge point A to edge point B.

Green's Theorem [67] is used to calculate the area enclosed by the cell. Green's Theorem states for a vector field $\boldsymbol{F}(x,y) = M(x,y)\boldsymbol{i} + N(x,y)\boldsymbol{j}$, and a region S bounded by a piecewise smooth, simple closed curve C,

$$\iint_S \left(\frac{\partial N}{\partial x} - \frac{\partial M}{\partial y}\right) dA = \oint_C M dx + N dy. \tag{4.8}$$

Note that for $\boldsymbol{n}$ as a unit normal to $\boldsymbol{F}$,

$$\begin{aligned}\oint_C \boldsymbol{F}\cdot\boldsymbol{n} ds &= \oint_C (M\boldsymbol{i} + N\boldsymbol{j})\cdot\left(\frac{dy}{ds}\boldsymbol{i} - \frac{dx}{ds}\boldsymbol{j}\right) dx \\ &= \oint_C -N dx + M dy \\ &= \iint_S \left(\frac{\partial M}{\partial x} - \frac{\partial N}{\partial y}\right) dA.\end{aligned}$$

If

$$\boldsymbol{F} = x\boldsymbol{i},$$

then it follows directly that

$$\boldsymbol{F} \cdot \boldsymbol{n} = x\boldsymbol{n}_x,$$

where $\boldsymbol{n}_x$ is the x component of the normal. From this, it can also be seen that

$$\begin{aligned}
\oint_C \boldsymbol{F} \cdot \boldsymbol{n} ds &= \oint_C -N dx + M dy \\
&= \oint_C x \\
&= \iint_S \frac{\partial x}{\partial x} dA \\
&= \iint_S dA \\
&= A(S).
\end{aligned}$$

For the simulation, S is the area enclosed in the cellular boundary. As the boundary is defined as a piecewise linear curve, the area of S can be calculated by

$$\iint_S dA = \sum_{i=1}^{L} \oint_{A_i}^{B_i} x\boldsymbol{n}_x d\boldsymbol{\tau} \tag{4.9}$$

In practice,

$$A(S) = \sum_{l=1}^{L} \frac{1}{2} |x_{B_l} - x_{A_l}| * \boldsymbol{n}_x(l) * |\boldsymbol{l}|, \tag{4.10}$$

where $\boldsymbol{n}_x(l)$ is the x component of the normal to edge l and $|\boldsymbol{l}|$ is the length of edge l. x_{B_l} and x_{A_l} are the x-coordinates of the two ends of edge l.

Recall from Chapter 4.2.3 that the nuclear boundary, is modeled as a rigid body. In practice, this implies that any forces acting on a single nuclear point must be applied to all nuclear points to ensure the cohesiveness of the nucleus.

Total Force

The final force acting on each point is the accumulation of all the forces acting that point. Thus, for any given point A, the total force acting on that point is given by

$$F_A = \sum_{B \in N(A)} F^s_{A,B} + 1_{[A \in \phi]} \left(\sum_{B \in N(A) \cap B \in \phi} F^p_{A,B} + F^s_{A,C} \right) + 1_{[A \in \mathcal{N}]} \sum_{B \in \mathcal{N} \setminus A} F_B, \quad (4.11)$$

where $N(A)$ is defined as all of the other points immediately connected to A. $1_{[x]}$ is the indicator of the logical predicate x, where

$$1_{[x]} = \begin{cases} 1 & x \text{ is true} \\ 0 & x \text{ is false.} \end{cases}$$

The point C is defined to be the calculated center of the nucleus, with ϕ as the boundary of the cell, and $\mathcal{N}$ is the set of points defining the boundary of the nucleus.

4.4.2 Integration Scheme

There are many numerical integration schemes available for use in solving ordinary differential equation initial value problems. Among the simplest is the Euler integration scheme, which solves the initial value problem

$$\frac{dy}{dt} = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha$$

by choosing a mesh along the t axis of granularity N. For each t_i in the time mesh, $w(t_i) \approx y(t_i)$, and

$$w(t_{i+1}) = w(t_i) + (t_{i+1} - t_i) f\big(t_i, w(t_i)\big). \quad (4.12)$$

However, the error in w grows linearly with Δt. [9]

Other explicit numerical integrators are more commonly in use, including the fourth-order Runge-Kutta method, the Runge-Kutta-Fehlberg method, and several multistep Adams techniques [9]. However, mass-spring differential equation systems are notoriously stiff [4, 51], meaning the timestep for explicit integration schemes must be small to prevent overwhelming errors in the results. The primary method of overcoming equation stiffness in computer graphics, where mass-spring physics simulations are in constant use, is the implicit Euler integration scheme. For the implicit Euler scheme, (4.12) is modified so that the new value is dependent on itself, as follows:

$$w(t_{i+1}) = w(t_i) + (t_{i+1} - t_i) f\big(t_{i+1}, w(t_{i+1})\big), \tag{4.13}$$

which produces a numerical integration scheme that is stable for all timesteps. As with the original Euler method, this implicit scheme is still $O(\Delta t)$, so its error grows linearly with Δt.

The simulation should obtain more accurate results than the implicit Euler method, so a second-order implicit method - the Implicit Trapezoidal method - is used. The Implicit Trapezoidal method is a fast extension to the Implicit Euler method, and is

given by

$$w(t_0) = \alpha$$
$$w(t_{i+1}) = w(t_i) + \frac{1}{2}(t_{i+1} - t_i)\Big(f\big(t_{i+1}, w(t_{i+1})\big) + f\big(t_i, w(t_i)\big)\Big),$$

where α is the initial value to the solution of the differential equation. The Implicit Trapezoidal method yields a balance between the step size range, the accuracy of the solution, and the speed of the integration method.

Implementation

At each timestep, the mesh updates consist of the solution to a massive system of differential equations based on Newton's laws of physics. At each node, the force vector acting on that node and its current position are known. The goal is to find the position of that node at the next timestep. From basic Newtonian mechanics,

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{v}(t),$$
$$\frac{d\boldsymbol{v}}{dt} = \boldsymbol{a}(t), \text{ and}$$
$$\boldsymbol{F}(t) = m\boldsymbol{a}(t),$$

where $\boldsymbol{x}(t)$ is the position of a node at time t, $\boldsymbol{v}(t)$ is the velocity of the node at time t, $\boldsymbol{a}(t)$ is the acceleration of the node at time t, $\boldsymbol{F}(t)$ is the total force impulse acting on the node at time t, and m is the mass of the node. However, these laws of motion

can be combined into two equations. Allowing $\frac{d}{dt}f(\cdot) = f'(\cdot)$,

$$\boldsymbol{v}'(t) = \frac{\boldsymbol{F}(t)}{m}$$

$$\boldsymbol{x}'(t) = \boldsymbol{v}(t).$$

For each node i, it can be calculated by the Implicit Trapezoidal method that

$$\boldsymbol{v}_i(t+\Delta t) = \boldsymbol{v}_i(t) + \frac{1}{2}(\Delta t)\Big(\boldsymbol{F}_i\big(\boldsymbol{x}_i(t+\Delta t)\big) + \boldsymbol{F}_i\big(\boldsymbol{x}_i(t)\big)\Big)/m_i \tag{4.14}$$

$$\boldsymbol{x}_i(t+\Delta t) = \boldsymbol{x}_i(t) + \frac{1}{2}(\Delta t)\big(\boldsymbol{v}_i(t+\Delta t) + \boldsymbol{v}_i(t)\big). \tag{4.15}$$

There are several assumptions in play for (4.15). The first is an assumption of a frictionless surface. It is also assumed the springs in the system are undamped. These assumptions indicate the force acting on each node is independent of its velocity.

It is possible to obtaining a closed-form solution for $\boldsymbol{v}_i(t+\Delta t)$. Following the basics given by Müller [51],

$$\begin{aligned} m_i\boldsymbol{v}_i(t+\Delta t) &= m_i\boldsymbol{v}_i(t) + \frac{1}{2}\Delta t\boldsymbol{F}_i\big(\boldsymbol{x}_i(t+\Delta t)\big) + \frac{1}{2}\Delta t\boldsymbol{F}_i\big(\boldsymbol{x}_i(t)\big) \\ &= m_i\boldsymbol{v}_i(t) + \frac{1}{2}\Delta t\boldsymbol{F}_i\left(\boldsymbol{x}_i(t) + \frac{1}{2}\boldsymbol{v}_i(t+\Delta t) + \frac{1}{2}\boldsymbol{v}(t)\right) + \frac{1}{2}\Delta t\boldsymbol{F}_i\big(\boldsymbol{x}_i(t)\big) \end{aligned}$$

By taking the Taylor expansion of $\boldsymbol{F}_i(\cdot)$ around $\boldsymbol{x}_i(t)$,

$$\begin{aligned} m_i\boldsymbol{v}_i(t+\Delta t) = m_i\boldsymbol{v}_i(t) &+ \frac{1}{2}\Delta t\boldsymbol{F}_i\big(\boldsymbol{x}_i(t)\big) + \frac{1}{4}\Delta t^2\big(\boldsymbol{v}_i(t+\Delta t) + \boldsymbol{v}_i(t)\big)\boldsymbol{F}_i'\big(\boldsymbol{x}_i(t)\big) \\ &+ \frac{1}{2}\Delta t\boldsymbol{F}_i\big(\boldsymbol{x}_i(t)\big) \end{aligned}$$

Now, if $\boldsymbol{K}_i$ is defined to be the Jacobian of $\boldsymbol{F}_i$, that is $\boldsymbol{K}_i = \boldsymbol{F}_i'$, then

$$m_i\boldsymbol{v}_i(t+\Delta t) = m_i\boldsymbol{v}_i(t) + \Delta t\boldsymbol{F}_i\big(x_i(t)\big) + \frac{1}{4}\Delta t^2\boldsymbol{K}_i\boldsymbol{v}_i(t+\Delta t) + \frac{1}{4}\Delta t^2\boldsymbol{K}_i\boldsymbol{v}_i(t)$$

$$-\frac{1}{4}\Delta t^2\boldsymbol{K}_i\boldsymbol{v}_i(t+\Delta t) + m_i\boldsymbol{v}_i(t+\Delta t) = m_i\boldsymbol{v}_i(t) + \Delta t\boldsymbol{F}_i\big(x_i(t)\big) + \frac{1}{4}\Delta t^2\boldsymbol{K}_i\boldsymbol{v}_i(t)$$

$$\left(m_i - \frac{1}{4}\Delta t^2\boldsymbol{K}_i\right)\boldsymbol{v}_i(t+\Delta t) = m_i\boldsymbol{v}_i(t) + \Delta t\boldsymbol{F}_i\big(x_i(t)\big) + \frac{1}{4}\Delta t^2\boldsymbol{K}_i\boldsymbol{v}_i(t) \quad (4.16)$$

If $\boldsymbol{K}$ can be calculated, then (4.16) can be solved by any number of methods. Note that for the simulation, $\boldsymbol{K}$ is a $2n \times 2n$ matrix, where n is the number of points in the mesh. Hence

$$\boldsymbol{K} = \begin{bmatrix} \frac{\partial F_x(n_1)}{\partial x_1} & \frac{\partial F_x(n_1)}{\partial y_1} \cdots \\ \frac{\partial F_y(n_1)}{\partial x_1} & \frac{\partial F_y(n_1)}{\partial y_1} \cdots \\ \vdots & \ddots \quad \vdots \\ \frac{\partial F_y(n_N)}{\partial x_1} & \frac{\partial F_y(n_N)}{\partial y_1} \cdots \end{bmatrix}.$$

Recall from (4.11) that the total force on a node A is given by

$$F_A = \sum_{B \in N(A)} F^s_{A,B} + 1_{[A\in\phi]}\left(\sum_{B\in N(A)\cap B\in\phi} F^p_{A,B} + F^s_{A,C}\right) + 1_{[A\in\mathcal{N}]}\sum_{B\in\mathcal{N}\setminus A} F_B.$$

It can be shown (see Appendix A.1) that the 2×2 submatrix $K_{A,X}$ of the Jacobian

of F_A is given by

$$\begin{aligned}
\boldsymbol{K}_{A,X} = {} & 1_{[X\in N(A)]}k_s\left(-\boldsymbol{I} - \frac{l_0(A,X)}{l(A,X)}\left(\boldsymbol{I} - \frac{(\boldsymbol{x}_X - \boldsymbol{x}_A)(\boldsymbol{x}_X - \boldsymbol{x}_A)^T}{l(A,X)^2}\right)\right) \\
& + 1_{[A\in\phi]}k_s\left(-\boldsymbol{I} - \frac{l_0(A,C)}{l(A,C)}\left(\boldsymbol{I} - \frac{(\boldsymbol{x}_C - \boldsymbol{x}_A)(\boldsymbol{x}_C - \boldsymbol{x}_A)^T}{l(A,C)^2}\right)\right) \\
& + 1_{[A\in\phi]}1_{[X\in\phi]}\left(-\frac{k_p}{A^2}\begin{pmatrix} y_Z - y_Y \\ x_Y - x_Z \end{pmatrix}(\nabla_X A)^T\right) \\
& + 1_{[A\in\phi]}1_{[X==Y]}\left(k_p\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\left(\frac{1}{A} - \frac{1}{A_0}\right)\right) \\
& - 1_{[A\in\phi]}1_{[X==Z]}\left(k_p\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\left(\frac{1}{A} - \frac{1}{A_0}\right)\right),
\end{aligned} \tag{4.17}$$

where y_X is the y-coordinate of node X, x_X is the x-coordinate of node X, and $\boldsymbol{x}_X$ is the two-dimensional coordinate vector for node X. Also, Y is defined to be the next boundary point from A in a clockwise direction, and Z is the previous point from A in a clockwise direction.

Let Y and Z be defined as above. Then the gradient of A with respect to the boundary point X is

$$\begin{aligned}
\frac{\partial A}{\partial x_X} &= -\frac{|x_Y - x_X|}{2(x_Y - x_X)}(y_X - y_Y) + \frac{|x_X - x_Z|}{2(x_X - x_Z)}(y_Z - y_X) \\
\frac{\partial A}{\partial y_X} &= \frac{1}{2}|x_Y - x_X| - \frac{1}{2}|x_X - x_Z|.
\end{aligned}$$

The implementation of (4.17) is surprisingly simple. For each node A, the first sum is calculated by cycling around all the springs connected to that node. The boundary nodes are ordered clockwise, and the resulting order is used to calculate

the gradient of A. The boundary nodes are used to calculate the remaining terms in (4.17). It should be noted that nuclear nodes are handled as standard nodes for this calculation.

If $\boldsymbol{V}(t)$ is defined to be the vectorization of $\boldsymbol{v}_i(t)$ for all i, then (4.16) can be written as

$$\left(\boldsymbol{m}\boldsymbol{I}_{2n\times 2n} - \frac{1}{4}\Delta t^2 \boldsymbol{K}\right)\boldsymbol{V}(t+\Delta t) = \boldsymbol{m}\boldsymbol{I}_{2n\times 2n}\boldsymbol{V}(t) + \Delta t\boldsymbol{F}(\boldsymbol{x}(\boldsymbol{t})) + \frac{1}{4}\Delta t\boldsymbol{K}\boldsymbol{V}(t), \quad (4.18)$$

where $\boldsymbol{m}$ is a vector of size $2n$ generated by repeated values of the node mass, such as

$$\boldsymbol{m} = (m_1, m_1, m_2, m_2, \ldots, m_n, m_n)^T.$$

After building $\boldsymbol{K}$, the solution of (4.16) can be obtained by conjugate gradient methods. Once $\boldsymbol{V}(t+\Delta t)$, has been calculated, (4.15) is used to obtain the new positions of each node.

4.4.3 Constraint Checks

There are several constraints in the simulation that must be fulfilled for each time step. First, the maximum distance a single point can move in a single time step must be constrained. This constraint is dependent on the position of each node and its neighbors. By constraining the maximum movement distance, several issues can be relieved, such as artifically artificially inflated velocity acting on border nodes and the resulting mesh failure. A second reason to constrain the node velocity is to prevent mesh "tangling," a condition in which mesh edges intersect in locations other than

common nodes.

Second, cells are prevented from overlapping. This constraint is for computational and tracking reasons, as it is possible *in vitro* for the lamellipodium of one cell to slide underneath the body of a second cell. Finally, the nucleus is constrained to stay within the cytoplasmic boundaries. Enforcing this final constraint also ensures that the nodes of the nucleus stay in constant positions with respect to each other.

Velocity and Node Movement Limitation

The maximum velocity of a given cellular node must be limited due to the possibility of numerical instability in the solution of (4.18). The velocity is limited after calculating the solution of (4.18). To implement this constraint, the longest edge in the mesh is located, and this length is scaled by $0.75/h$ to find a maximum velocity threshold, where h is the time difference between steps in the simulation. The magnitude of the velocity for each node is set to the lesser of the maximum velocity threshold and the calculated velocity magnitude.

To limit node movement and prevent mesh tangling, ever potential node movement is tested in sequence. Intersections between edges connected to the moving node and any other edges in the mesh are determined for each node. If there are any intersections detected, then instead of the calculated position, the new node position is set to be the weighted average of the positions of surrounding nodes weighted by the current resting length of the edge between the node being moved and the appropriate

connected node.

Overlap prevention

To prevent overlap between the cells, a standard collision detection scheme, as first implemented by Walaber [70], is implemented. Under this scheme, axis-aligned bounding boxes are compared as an initial step to search for overlapping cells. If the bounding boxes of cells i and j overlap, all the nodes along the boundary of i are checked to see if they are contained within j and vice versa. If a node A from cell i is contained within cell j, the nodes B_1 and B_2 which define the boundary edge closest to A are determined. Figure 4.6 shows an example of a collision with node A from cell i being inside the boundary of cell j, with the edge $\overline{B_1B_2}$ being the closest edge. From Newtonian physics, the change in the position of A, B_1, and B_2 can be calculated as

$$d\boldsymbol{x}_A = \boldsymbol{N}\frac{p}{m_{B_1} + m_{B_2}}, \tag{4.19}$$

$$d\boldsymbol{x}_{B_1} = -\boldsymbol{N}\left(\frac{p}{m_A}\right)\left(\frac{d_1}{||\boldsymbol{x}_{B_1} - \boldsymbol{x}_{B_2}||}\right), \text{ and} \tag{4.20}$$

$$d\boldsymbol{x}_{B_2} = -\boldsymbol{N}\left(\frac{p}{m_A}\right)\left(1 - \frac{d_1}{||\boldsymbol{x}_{B_1} - \boldsymbol{x}_{B_2}||}\right). \tag{4.21}$$

Similarly, if the relative velocity $\boldsymbol{rV}$ between A and the edge $\overline{B_1B_2}$ is defined as

$$\boldsymbol{rV} = \boldsymbol{v}_A - \frac{1}{2}(\boldsymbol{v}_{B_1} + \boldsymbol{v}_{B_2}),$$

then the velocity multiplier J can be calculated as

$$J = \frac{0.5 * \boldsymbol{rV} \cdot \boldsymbol{N}}{\frac{1}{m_A} + \frac{1}{m_{B_1} + m_{B_2}}}.$$

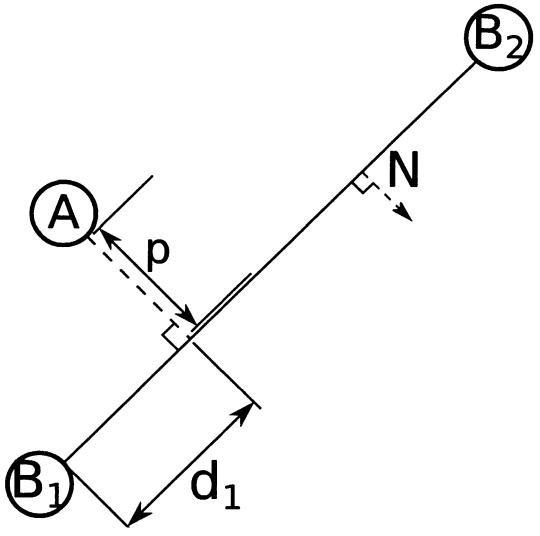


Figure 4.6 : Diagram of intersection of two cells. Node A is inside the boundary of the cell with boundary points B_1 and B_2, where the edge $\overline{B_1B_2}$ is the closest to A. $\boldsymbol{N}$ is the outward unit normal to $\overline{B_1B_2}$. p is the orthogonal distance from A to $\overline{B_1B_2}$, and d_1 is the distance from node B_1 to the projection of A onto $\overline{B_1B_2}$.

Using the above equations, the change in velocity for nodes A, B_1, and B_2 can be calculated as as

$$d\boldsymbol{v}_A = \frac{J\boldsymbol{N}}{m_A}, \tag{4.22}$$

$$d\boldsymbol{v}_{B_1} = -\frac{J\boldsymbol{N}}{m_{B_1} + m_{B_2}} \left(\frac{d_1}{||\boldsymbol{x}_{B_1} - \boldsymbol{x}_{B_2}||} \right), \text{ and} \tag{4.23}$$

$$d\boldsymbol{v}_{B_2} = -\frac{J\boldsymbol{N}}{m_{B_1} + m_{B_2}} \left(1 - \frac{d_1}{||\boldsymbol{x}_{B_1} - \boldsymbol{x}_{B_2}||} \right). \tag{4.24}$$

Nuclear cohesion

The first step to ensuring the cohesion of the nucleus is to force all the nodes on the nuclear boundary to move with the same velocity. This constraint is satisfied by setting the velocity of every nuclear boundary nodes to the average of all the nuclear boundary nodes before the positions are updated, ensuring the nodes of the nuclear boundary always move at the same speed.

The second step is to prevent the nucleus from moving outside the cytoplasmic boundary, which is accomplished through a modification of the collision detection algorithm used to prevent the different cell from overlapping. The collision detection method tests the nuclear boundary points to see if they fall outside the area of the cytoplasmic boundary. Figure 4.6 depicts this situation as well, except with A being outside the boundary defined by $\overline{B_1 B_2}$, and $\boldsymbol{N}$ being the inward unit normal. Equations (4.19)- (4.24) are used to update the positions and velocities of A, B_1, and B_2. It must be noted that in order for the nucleus to retain its shape, the positions and velocities of all the nuclear vertices must updated through (4.19) and (4.22).

4.5 Simulating Protein Concentration

The oscillatory behavior of the simulation intensities is based on a damped sinusoidal curve. Each cell is assigned a minimum intensity mean and standard deviation (μ_l and σ_l) and a maximum intensity mean and standard deviation (μ_h and σ_h). The distributions for the nuclear and cytoplasmic intensities in each frame are determined through the following equations:

$$u = \frac{1}{2 + t/32} \cos(b * t + \pi) + \frac{1}{2 + t/32}, \tag{4.25}$$

$$\text{cytoplasm} \sim (1 - u)N(\mu_h, \sigma_h^2) + uN(\mu_l, \sigma_l^2), \text{ and} \tag{4.26}$$

$$\text{nucleus} \sim uN(\mu_h, \sigma_h^2) + (1 - u)N(\mu_l, \sigma_l^2), \tag{4.27}$$

where b is a multiplier to adjust the frequency of oscillation.

To generate the data, the background of the image is first filled with independent exponential variates ($\mu = 1$). Each cell is examined, and the value of u is determined, where t is the current time step of the simulation. Every pixel in the bounding box of the cell is examined. If the given pixel is within the cytoplasmic boundary, it is assigned a pixel intensity value of an indpendent random normal deviate from (4.26). If the random deviate is greater than 255, the pixel intensity is set to 255, and if the deviate is less than 0, it is the intensity is set to 0. A similar procuedure is performed using the bounding box for the nucleus, with the pixel intensities drawn indpendently from (4.27).

The three-step procedure described here yields an image where the nuclear and

cytoplasmic compartments of each cell exhibit oscillatory behavior in their mean intensities, and independent normally distributed error in the cytoplasmic and nuclear distributions.

The simulation as described in the chapter allows for testing of the tracking algorithm against time series where the intensity distributions of the cytoplasm and nuclear areas can be either very similar or exactly the same.

Chapter 5

Results

5.1 Simulation Output

We attempted to run 400 simulations with a variety of parameters to test both the simulation mechanism and the proposed tracking algorithm. Chapter 5.1.1 describes the different parameters varied as well as describing the most noticeable effects of varying the different parameters. Chapter 5.1.2 describes several parameter combinations, as well as providing both mesh and filled output for each simulation discussed.

5.1.1 Parameter Variations

The parameters varied in each run are the number of corners of the regular polygon used to generate the cell, the radius of the polygon used to generate the cell, the percentage of the cell area used by the nucleus, the speed of cell movement, the speed of protein translocation, and the concentration of cells in the simulation. The majority of the parameters are varied over three possibilities, with the exception of the number of corners in the cell. The descriptions for each of the parameter variants can be found in Tables 5.1-5.6. Initial testing showed that any polygon with less than 5 corners had a high likelihood of self-intersection in the amorphic cytoplasm

	Number of Corners
1	3
2	5
3	10
4	25
5	50

Table 5.1 : Number of corners allowed in the simulation

	Cell Size	Minimum Radius (pixels)	Maximum Radius (pixels)
1	small	30	50
2	medium	50	90
3	large	90	130

Table 5.2 : Minimum and maximum radii for different sizes of cells

boundary, and could therefore not generate cells at all. Therefore, for the generation of our simulation library, the simulation was limited to having only cells generated from regular polygons with 5, 10, 25, and 50 corners.

We examined the effect each parameter had on the stability of the simuation, as described by the number of images produced in the run. For all of our runs, we attempted to simulate the first 50 "seconds" with a time step of 0.01 "seconds." We

	Nuclear Size	Minimum Area	Maximum Area
1	small	5%	15%
2	medium	15%	25%
3	large	30%	40%

Table 5.3 : Minimum and maximum percentage of the cytoplasm covered by the nucleus

	Movement Speed	Minimum k_S	Maximum k_S	Time in state multiplier
1	slow	5	6	2
2	medium	9	11	1
3	fast	18	22	0.5

Table 5.4 : Descriptions of cell movement speed

	Oscillation Speed	Minimum period multiplier	Maximum period multiplier
1	slow	0.2	0.4
2	medium	0.4	0.6
3	fast	0.6	0.8

Table 5.5 : Minimum and maximum period multiplier for each speed of protein translocation.

	Density	Percent Coverage
1	sparse	25%
2	medium	50%
3	fast	75%

Table 5.6 : Estimated percent image coverage for each cell density possibility.

also attempt to output 300 images over this time course, although rounding errors yield a maximum of 312 images per simulation, with one image every 16 time steps.

Figure 5.1a shows the effect of the number of corners on simulation stability. It can be seen that having either too many or too few corners causes a decrease in the stability of the simulation. This is most likely due to a low number of corners yielding sudden sharp turns in the cytoplasmic boundary and large number of corners yielding a series of spikes along the cytoplasmic boundary.

Figure 5.1b shows the effect of cell size on the stability of the simulation. As expected, larger cells are far more stable, due to a minimal amount of corrections needing to be done to keep the meshes making up these cells from tangling.

Figure 5.1c shows that the nuclear size has little to no effect on simulation stability. The small tendency toward lower simulation lengths for larger nuclei is likely due to the increased likelihood of the nuclear boundary nearing the cytoplasmic boundary and causing algorithmic errors during a remeshing.

The cell concentration also has a significant effect on the stability of the simulation,

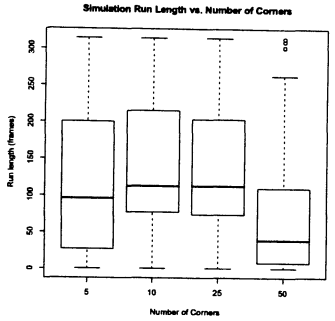
as can be seen in Figure 5.1d. This is due to two different factors. First, we make no efforts to correct for situation where the random placement of cells on the image does not allow for the total number of cells to be placed, which results in a simulation failure. Second, during our simulation, we only test for cytoplasmic intersection once per iteration. However, if two nuclei begin to overlap during the movement phase, the cytoplasmic boundaries will also be pushed to overlap. This can cause critical failure of the meshes in both cells, and failure of the simulation.

Figure 5.1e shows that an increase in movement speed and the corresponding increases in the basic spring constant has a detrimental effect on the stability of our simulation. This is due to the inherent fragility of mass-spring systems, which often take very precise tuning of constants to function properly to begin with. Similarly, a high rate of movement will allow more overlaps between cells and their corresponding nuclei, which allows more possibility of critical mesh failure.

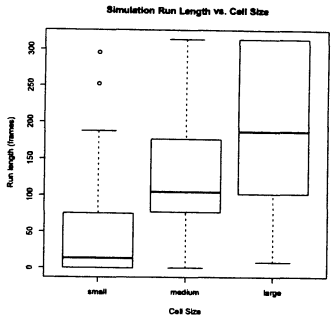
Figure 5.1f shows the lack of effect the protein translocation speed has on the stability of the simulation. This is to be expected, as the protein translocation speed is only used in the display of the cells, as opposed to the generation or movement of the cell meshes.

5.1.2 Example Results

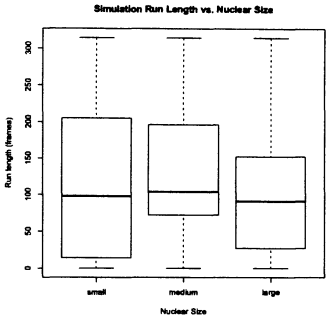
We built a library of simulations using 400 random combinations of the parameters discussed in Chapter 5.1.1. A simulation which succeeded in generating 50 or more



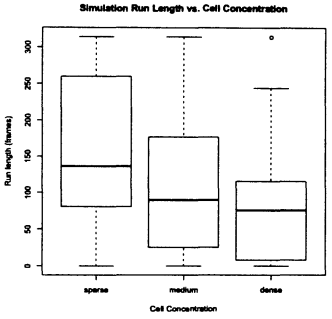
(a) Number of Corners



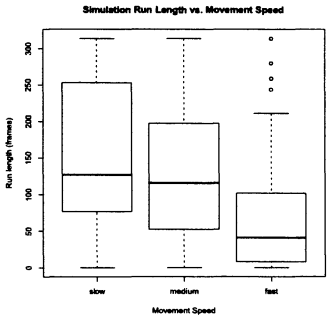
(b) Cell Size



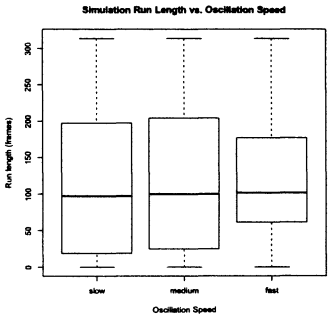
(c) Nuclear Size



(d) Cell Concentration



(e) Movement Speed



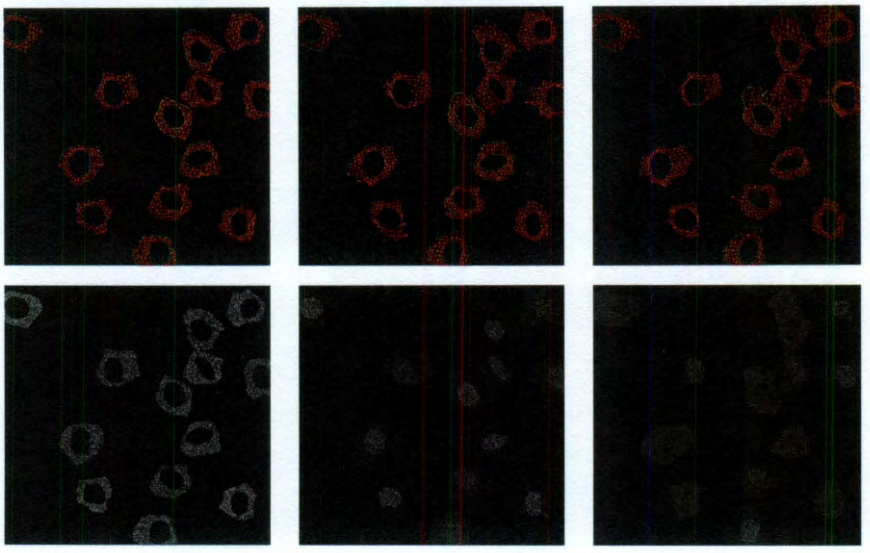
(f) Translocation Speed

Figure 5.1 : Effect of varying parameters on the simulation stability.

images (800 time increments) is considered to be trackable. Due to the influences mentioned above, only 279 parameter combinations resulted in what we consider to be trackable simulations.

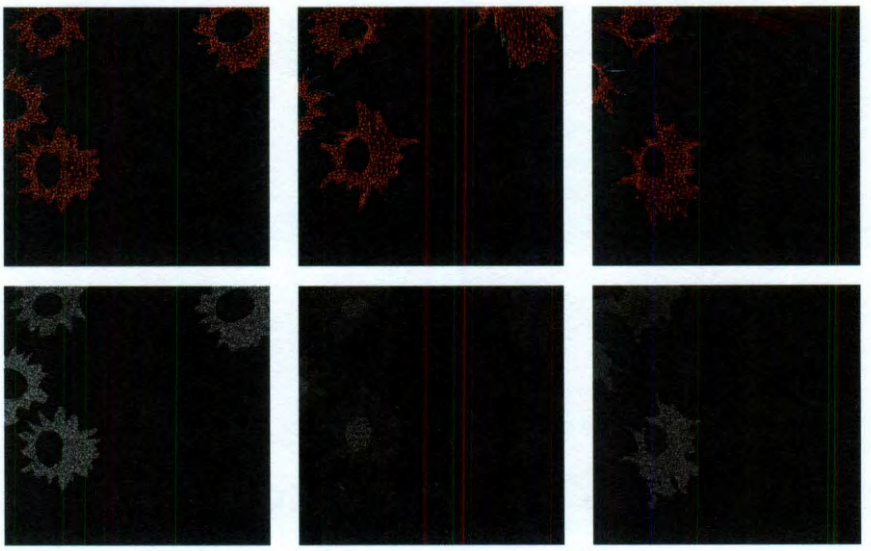
We discuss here three example simulations chosen from the 279 trackable simulations. These three simulations cover different choices of cell and nuclear size. The first example is generated using "small" cells and "large nuclei," i.e., the radius of the original polygon is between 30 and 50 pixels, and the nucleus covers between 30 and 40 percent of the cellular area. Figure 5.2 shows the mesh and resulting protein images from three time points in this simulation. Our second simulation is an example of a "medium" cell with a "medium" nuclear size. In this second example, of which select frames are shown in Figure 5.3, the radius of the original polygon is between 50 and 90 pixels, and the nucleus covers between 15 and 25 percent of the cellular area. Our final example simulation, shown in Figure 5.4 encompasses "large" cells (cellular boundaries generated from a polygon with radii between 90 and 130 pixels) and "small" nuclei, which consist of between 5 and 15 percent of the cellular area.

Figure 5.2 shows that our simulations successfully handles cells sliding past one another with some collision, although there is no head-on collision in this simulation. Although the final failure of the simulation did not occur until image 127 ($t = 20.32$), this simulation only produced 39 usable images. At image 40 ($t = 6.4$), there was a failure in one of the cell meshes, which destroyed both that cell and one other. Although the constraints on the simulation reduced the incidence of catastrophic



$t = 0$ $t = 3.04$ $t = 6.24$

Figure 5.2 : Time points from simulation 402. The simulation was run with parameters: **Corners**: 10, **Cell Size**: Small, **Nuclear Size**: Large, **Movement Speed**: Medium, **Translocation Speed**: Fast, **Cell Concentration**: Sparse

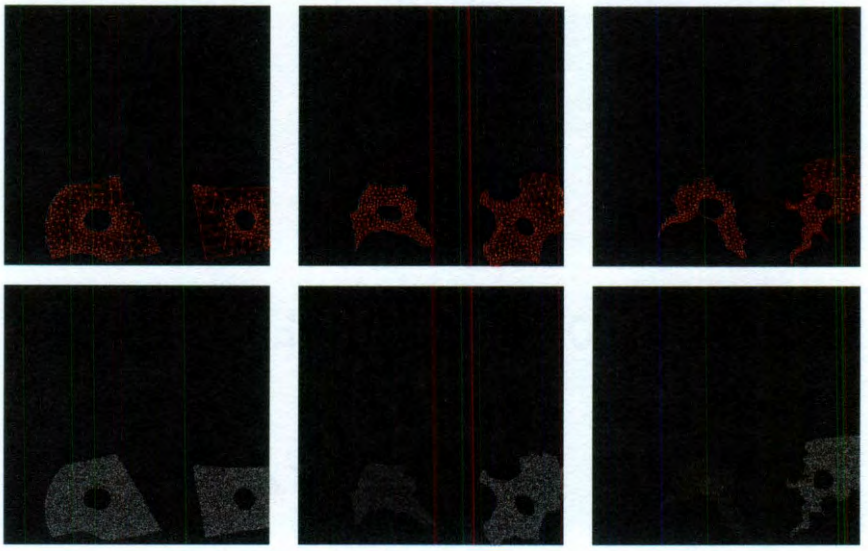


$t = 0$ $t = 14.08$ $t = 28.16$

Figure 5.3 : Time points from simulation 212. The simulation was run with parameters: **Corners**: 50, **Cell Size**: Medium, **Nuclear Size**: Medium, **Movement Speed**: Medium, **Translocation Speed**: Fast, **Cell Concentration**: Sparse

mesh failure, this simulation indicates that mesh failure can still occur under certain conditions.

Figure 5.3 shows well the simulation results of having a large number of corners in an image. Due to the uniform randomness in the polygon radius and the large number of points generated, the cells assume a roughly circular shape with a large number of spikes leading out in every direction. Although this yields a highly nonstandard cell shape, it is still a good test for our tracking algorithm. In addition, we can see that the cell in the upper-right corner of the image has suffered a critical failure sometime



$t = 0$ $t = 16.32$ $t = 32.48$

Figure 5.4 : Time points from simulation 130. The simulation was run with parameters: **Corners**: 5, **Cell Size**: Large, **Nuclear Size**: Small, **Movement Speed**: Fast, **Translocation Speed**: Slow, **Cell Concentration**: Sparse

before $t = 28.16$, which can be visually seen in the mesh image. However, this critical failure does not affect the other cells in the image, as it did in Simulation 402 (Figure 5.2).

Selections from our final simulation are shown in Figure 5.4. This simulation shows the results of having the minimum number of corners in the generating polygon. The resulting cellular boundaries obtain long, smooth lines in their initial configuration, which is easily seen in these larger cells due to the range of radii being between 90 and 130 pixels long. The extreme shrinking of the cell in the middle of the screen

is mostly due to the random cytoplasmic variation process, which seemed to have chosen contraction values more often than expansion values. This issue would best be resolved through parameter tuning.

5.2 Tracking Results

We tested the proposed algorithm one hundred times against every one of the 279 simulations mentioned in Chapter 5.1.2 and calculated the nine different error metrics described in Chapter 3.5. The percent distance error metric (Chapter 3.5.5) was chosen as the primary error metric for comparison against Cell Tracker. Of the 279 simulations, the best ten, middle ten, and worst ten simulations in terms of the maximum percent distance error metric were chosen to compare against naïve Cell Tracker runs.

The details of three selected simulations and the overall details of the 30 chosen simulations are detailed in Chapter 5.2.1. In addition, the proposed algorithm and Cell Tracker were both used on three sets of real data. A visual comparison of these results is included in Chapter 5.2.2. Finally, we tested the ability of the proposed algorithm to generate integrated intensity results compared to manual tracking data. These results are included in Chapter 5.2.3.

5.2.1 Simulation Tracking

We were able to successfully evaluate and accurately calcualte error metrics for 136 of the 279 trackable simulations. Although the simulations themselves ran correctly, some iterations of the proposed algorithm failed to track any cells (60 simulations), failed to complete a sufficient number of frames for a large number of the tracking runs (24 simulations), or incorrectly associated regions which were tracked correctly (59 simulations).

Figure 5.5 shows the normality of the percent distance error metric. As it was chosen to use the percent distance error metric as the base error metric for comparison between Cell Tracker and the proposed algorithm, it was necessary to determine the distribution of this error metric. Additionally, Figure 5.5 indicates that the average maximum percent distance error has greater than 50% probability of having some overlap with the actual nucleus.

We have chosen to examine select frames from three different simulations - one each from the low, middle, and high groups in terms of maximum percent distance error - for more detailed analysis.

Overall Comparison

To test the overall performance of the proposed algorithm against Cell Tracker, we examined each of the nine error metrics defined in Chapter 3.5. Our goal is to test two different hypotheses for each error metric. If we allow μ_p to be the average error

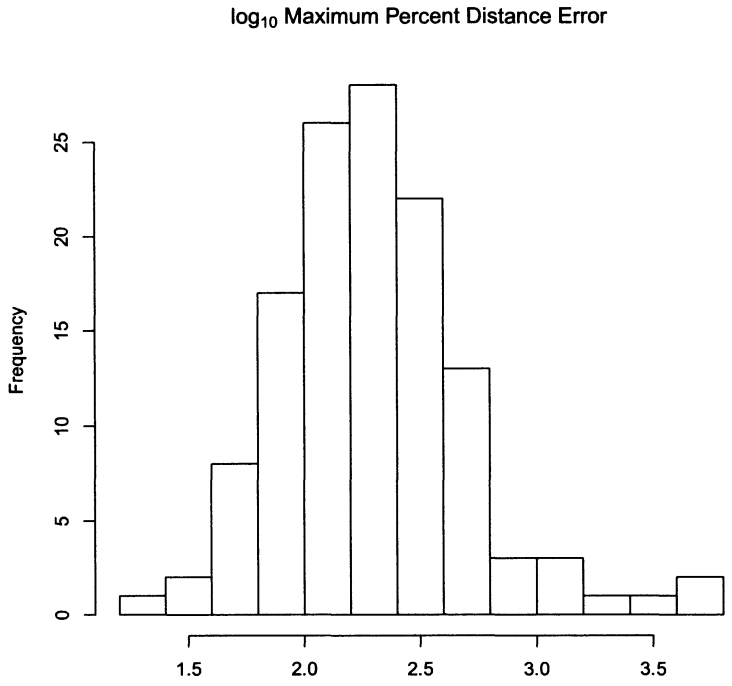


Figure 5.5 : Distribution of $\log_{10}$ of the maximum percent distance error for each simulation. For reference, a value of approximately 2.3 (200%) is a reasonable indication that the true position of the nucleus and the estimated position of the nucleus do not overlap.

metric for a given frame from the proposed algorithm and μ_c to be the error metric for a given frame generated by Cell Tracker, then we test the following hypotheses.

$$H_{0a} : \mu_p \geq \mu_c$$

$$H_{1a} : \mu_p < \mu_c$$

and

$$H_{0b} : \mu_p \leq \mu_c$$

$$H_{1b} : \mu_p > \mu_c$$

Because μ_p is an average error metric, we can assume that it has an underlying normal distribution by the Central Limit Theorem. The natural test statistic for a normally distributed random variable (μ_p) using an estimated variance is the t-statistic. Therefore, these two different hypotheses can be tested using a basic t-test. To determine the number of repetitions for the proposed algorithm, we chose to attempt to detect a relative difference in error metrics of 0.25. We also assumed a significance level for our tests of $\alpha = 0.05$ and a power of 0.8. From this, we calculate the needed sample size for our t-tests to be approximately 100 runs.

Now we allow p to be the number of frames where we can reject the H_{0a} at the $\alpha = 0.05$ significance level for a given error metric. Similarly, we set q to be the number of frames where we can reject H_{0b} at the $\alpha = 0.05$ significance level for the same error metric. We then claim that if $p > q$, then the proposed algorithm outperforms Cell Tracker with respect to that metric. Tables 5.7 and 5.8 show p and

Simulation	MIPCE	IMNCE	IMCCE	X	Y	Percent Distance	Rx	Ry	Rotation
121	75	75	62	75	75	75	9	0	16
125	83	66	83	56	76	71	66	26	31
139	59	55	59	59	54	57	30	27	13
140	50	33	50	50	28	50	50	49	36
168	186	22	13	81	113	98	123	30	71
173	108	43	106	34	25	49	88	88	74
184	52	15	52	35	28	6	9	1	33
207	72	72	72	57	48	60	67	71	28
218	202	101	202	177	174	173	151	178	143
227	96	102	89	102	102	102	102	102	28
231	237	65	237	93	68	46	112	158	46
280	104	14	104	20	42	15	74	29	38
295	314	190	314	280	157	211	12	92	198
328	189	51	193	78	90	40	22	40	118
329	113	26	113	84	66	62	52	32	39
344	177	44	176	87	129	82	93	70	75
350	71	27	69	16	16	19	47	34	38
366	153	11	153	50	72	72	83	13	99
369	12	28	61	64	63	43	15	0	52
371	23	28	75	57	45	28	0	0	37
375	127	60	127	77	110	60	24	29	61
383	162	101	162	96	108	136	155	49	82
391	96	50	96	84	69	34	47	4	61
395	76	21	77	37	55	4	25	2	58
400	83	15	73	22	15	7	75	12	88
411	101	75	82	76	76	73	53	52	60
443	123	0	128	16	42	0	0	0	67
453	136	96	136	136	124	128	110	85	68
477	146	61	154	105	123	115	83	72	67
500	91	8	84	49	68	20	78	46	37

Table 5.7 : The number of frames where the proposed algorithm has a lower mean error than Cell Tracker (detected at $\alpha = 0.05$ significance level) for each simulation and each error metric.

q, respectively, for each simulation.

Table 5.9 shows the number of error metrics where the proposed algorithm outperforms Cell Tracker for each simulation. For any simulation where the proposed algorithm outperforms Cell Tracker for more than 4 error metrics, we conclude that the proposed algorithm outperforms Cell Tracker for the entire simulation. We can see from Table 5.9 that the proposed algorithm outperforms Cell Tracker for 23 of the 30 simulations tested against both possibilities.

Simulation	MIPCE	IMNCE	IMCCE	X	Y	Percent Distance	Rx	Ry	Rotation
121	0	0	11	0	0	0	65	75	46
125	0	17	0	26	4	10	12	56	34
139	0	4	0	0	5	2	22	15	41
140	0	16	0	0	22	0	0	1	12
168	0	149	169	87	53	66	51	148	81
173	0	62	1	64	71	37	16	11	28
184	0	37	0	17	22	44	38	49	16
207	0	0	0	15	22	12	5	1	40
218	0	97	0	24	27	28	50	21	42
227	6	0	13	0	0	0	0	0	64
231	0	169	0	139	162	182	116	57	174
280	0	88	0	75	57	86	27	62	56
295	0	117	0	20	114	78	295	198	83
328	6	135	1	110	94	153	165	149	67
329	0	79	0	16	37	43	56	75	59
344	0	110	1	59	36	69	68	97	78
350	0	38	0	49	53	50	16	32	28
366	0	138	0	83	65	68	61	130	40
369	52	35	4	0	1	12	46	65	12
371	50	45	0	17	27	46	75	75	32
375	0	64	0	42	15	56	93	94	54
383	0	54	0	60	50	24	3	108	63
391	0	42	0	7	23	53	43	92	34
395	4	57	2	33	21	76	52	78	16
400	20	90	30	81	90	99	26	92	13
411	0	25	16	20	19	27	38	43	33
443	4	128	0	104	66	127	128	127	49
453	0	39	0	0	9	7	22	51	60
477	9	88	1	42	20	37	63	61	69
500	0	83	7	39	19	66	10	43	53

Table 5.8 : The number of frames where the proposed algorithm has a higher mean error than Cell Tracker (detected at $\alpha = 0.05$ significance level) for each simulation and each error metric.

Simulation	Proposed Algorithm	Cell Tracker
121	6	3
125	7	2
139	8	1
140	9	0
168	4	5
173	6	3
184	5	4
207	8	1
218	9	0
227	8	1
231	3	6
280	3	6
295	7	2
328	3	6
329	5	4
344	6	3
350	5	4
366	6	3
369	5	4
371	4	5
375	6	3
383	8	1
391	7	2
395	5	4
400	4	5
411	9	0
443	3	6
453	9	0
477	7	2
500	6	3

Table 5.9 : Number of error metrics where the proposed algorithm outperforms Cell Tracker and vice versa for each simulation. We consider one algorithm to outperform the other overall when it outperforms on five or more error metrics.

Low Maximum Percent Distance Error

Simulation 411 was generated using a sparse collection of small cells with large nuclei. The cells are generated from regular pentagons where slow movement and fast translocation are assumed. The simulation lasted for roughly 1600 iterations before failure. Select frames from this simulation are shown in Figure 5.6.

First examine Figure 5.7. Due to the odd habit in the simulation of opposite movement directions being used sequentially, it does not appear that the cells have moved a great deal in this image, which certainly assists in the tracking of the cellular and nuclear boundaries. The cell in the lower left hand corner of the image frame that is completely red remains untracked for some reason. The cell in the lower right corner of the image that drops to nothing but a nucleus on the final line is due to the loss of external boundary edges during the mesh decimation step. However, we can see that the proposed algorithm handles this issue correctly.

The fastest way to see obvious errors in the nuclear tracking results is to look for arcs of different color around the edges of the nuclei themselves. One of these can be seen in the rightmost image on the third line. The cell with the green nucleus and brownish cytoplasm near the right-hand side shows an area where the tracked nuclear region spills out into the higher intensity cytoplasm. In our tests, we found that the maximum average percent distance error for any given tracking run is 50.6%, indicating a significant continued overlap between the true location of the nuclear region and the tracking result. It must be noted, however, that the small size of the

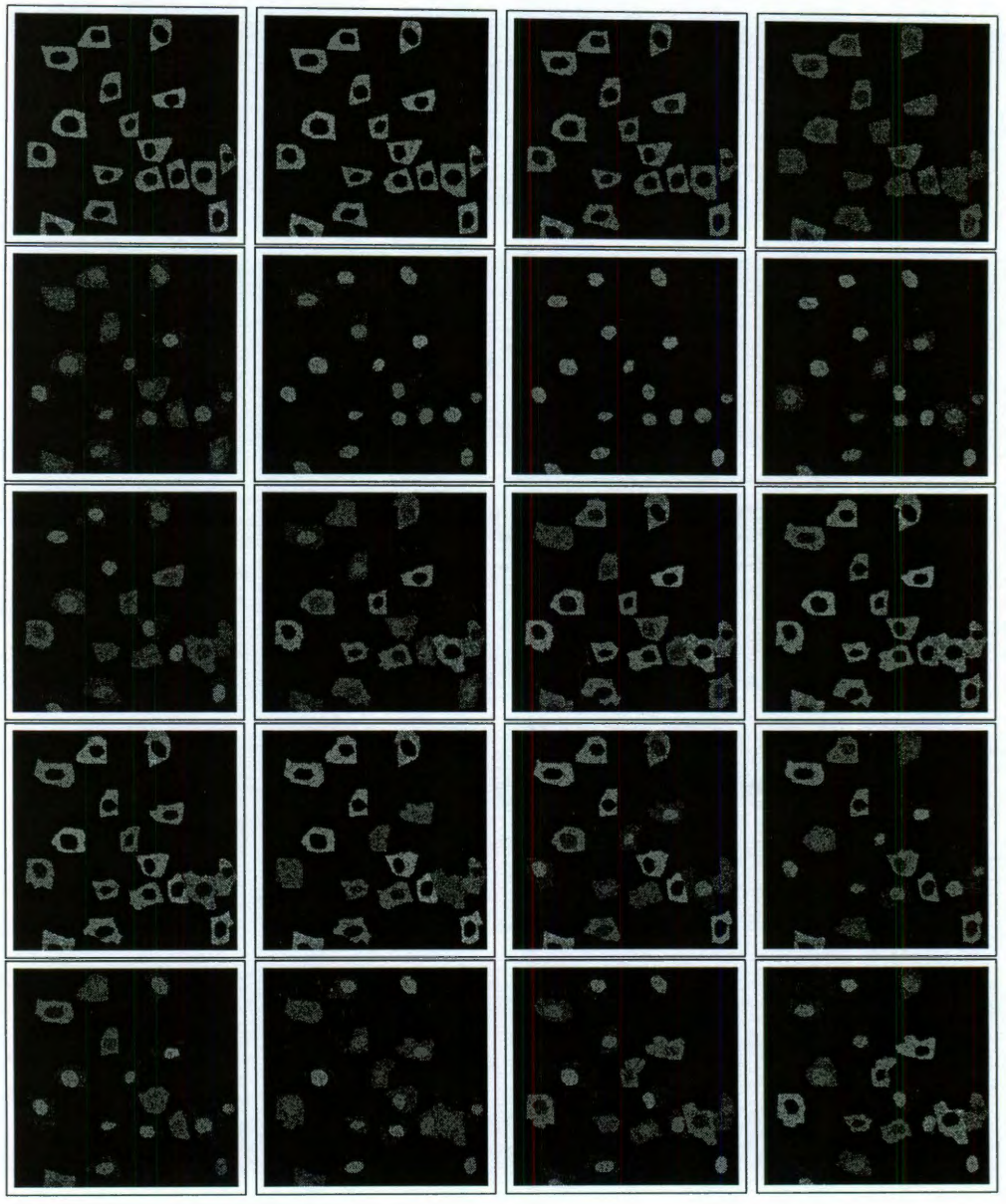


Figure 5.6 : Raw data from simulation 411 (Parameter details in text)

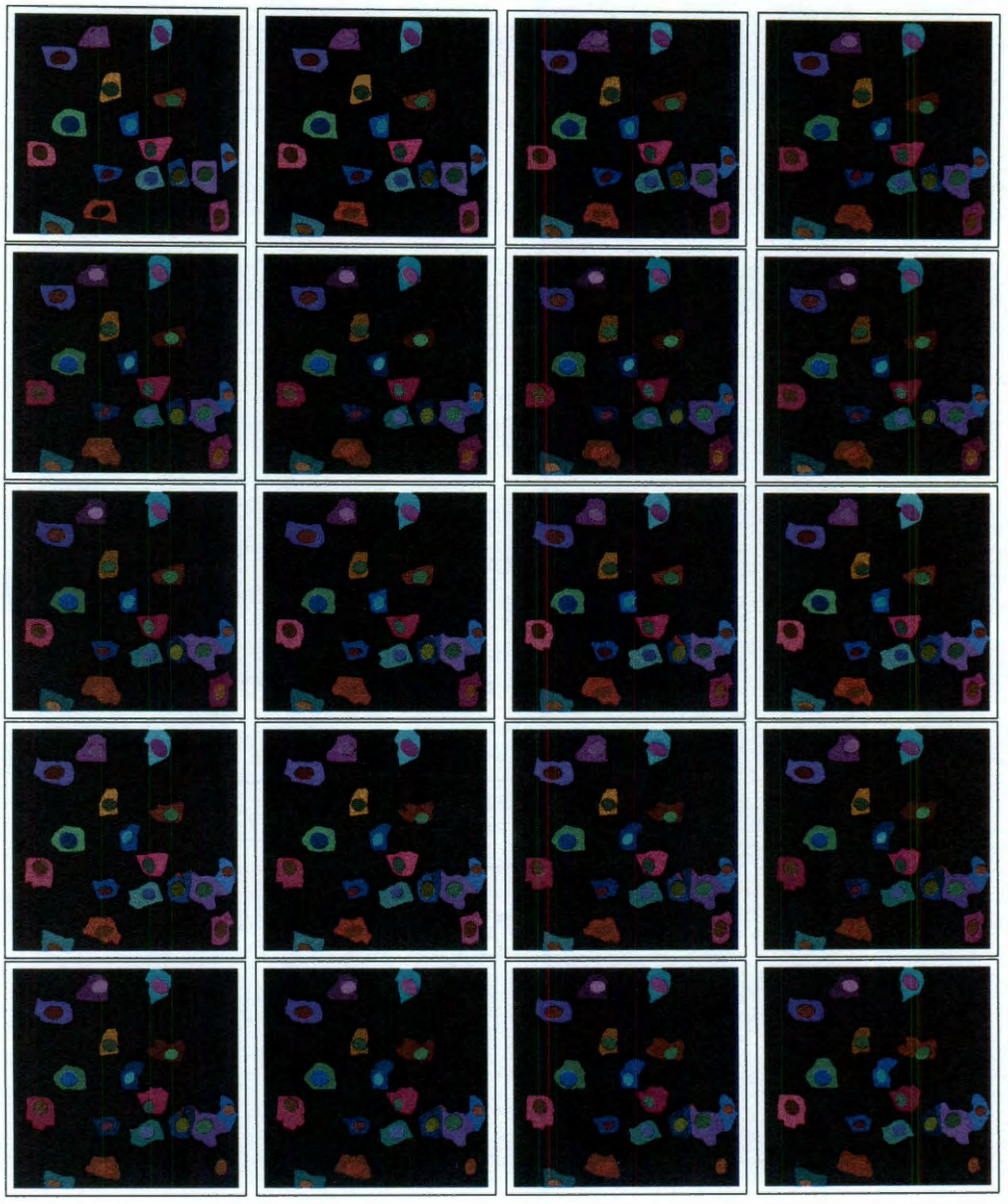


Figure 5.7 : Visualized tracking results from simulation 411 using the proposed algorithm. The regions are overlaid on the raw data from Figure 5.6.

nuclear areas will inflate the error values much faster than they would be for large nuclei in large cells. It would not be difficult to conceive that missing the true center of the nucleus by only 5 pixels may cause this large of an error.

Cell Tracker has trouble tracking some of these cells. This can be seen in Figure 5.8, where the nuclear boundary stops following the nuclear edge and instead is drawn to the cytoplasmic edge. One example of this is the reightmost cell in the clump of three cells near the bottom right corner of the images. Other than the problem of the nuclear boundary being drawn to the cytoplasmic edges, Cell Tracker does a reasonable job of tracking these cells.

Figures 5.9-5.17 show the error metrics for Cell Tracker and the mean error metrics for 100 tracking runs for this data, as well as the 95% confidence interval for the mean. Figures 5.10-5.17 also show the average standard deviation between cells in each frame and the 95% confidence interval for this average. Finally, Figure 5.17 shows that although we do normally observe the same error patterns as in Cell Tracker, the proposed algorithm is more accurate in the placement and orientation of the nuclear boundaries.

Figures 5.18-5.23 are a visual depiction of how well the proposed algorithm and Cell Tracker were able to calculate the integrated region intensity for the nuclear and cytoplasmic regions of select cells in the image. It is clear that for the majority of cells shown, the proposed algorithm more accurately calculates the integrated region intensities than Cell Tracker.

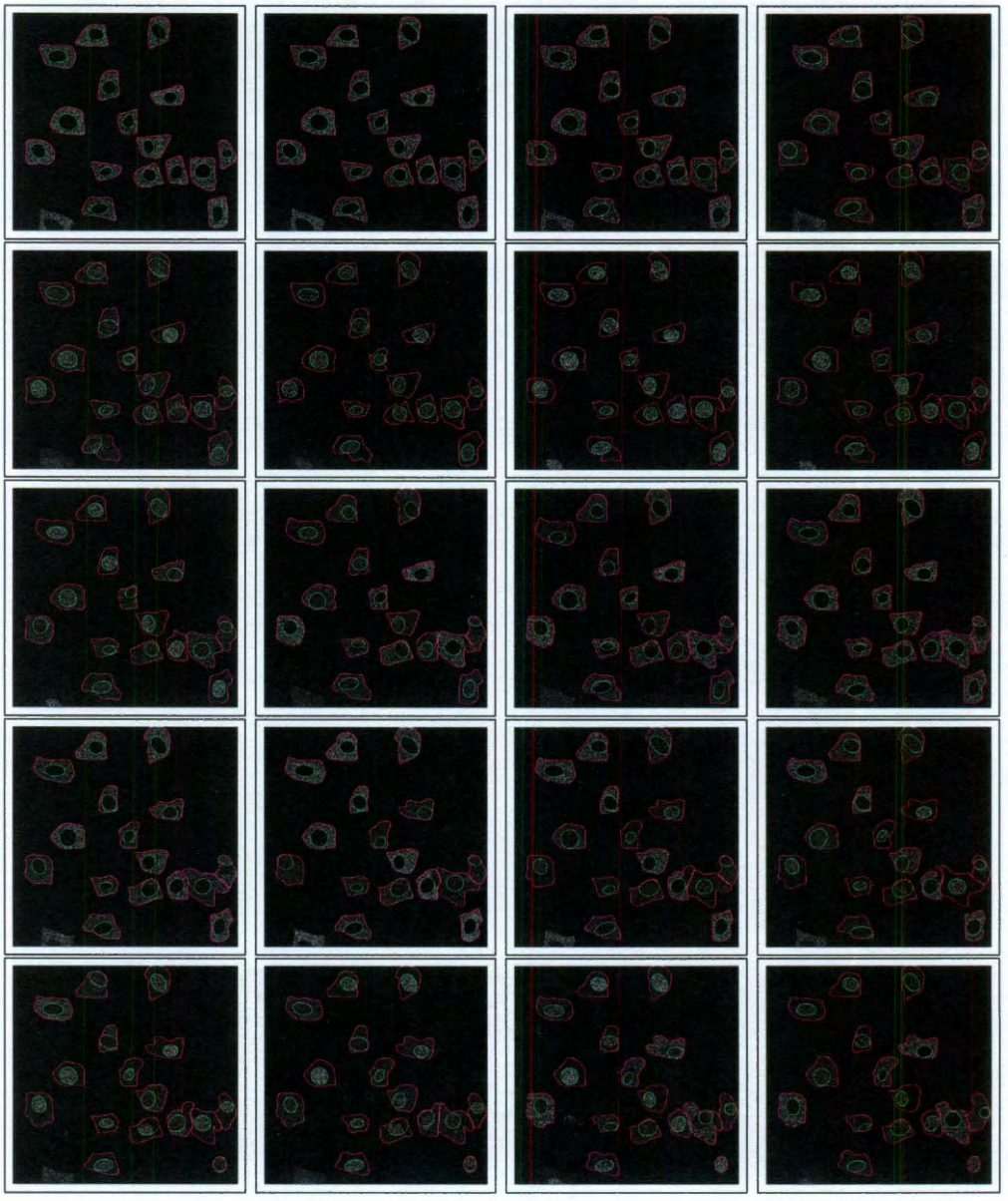


Figure 5.8 : Visualized tracking results from simulation 411 using Cell Tracker. The region boundaries are overlaid on the raw data from Figure 5.6.

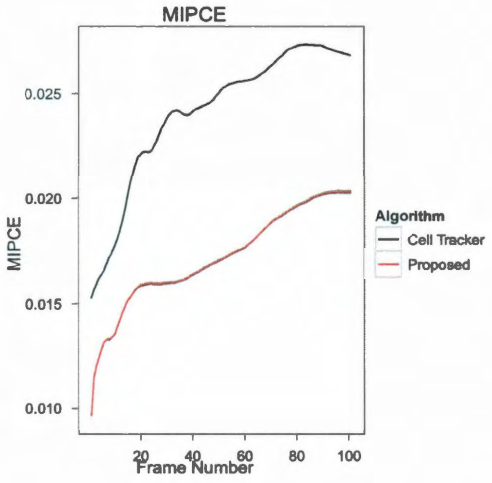


Figure 5.9 : Mean Integrated Percent Classification Error for simulation 411.

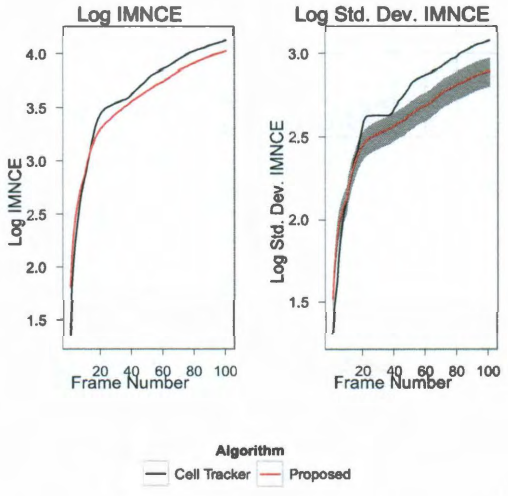


Figure 5.10 : Integrated Mean Nuclear Classification Error for simulation 411.

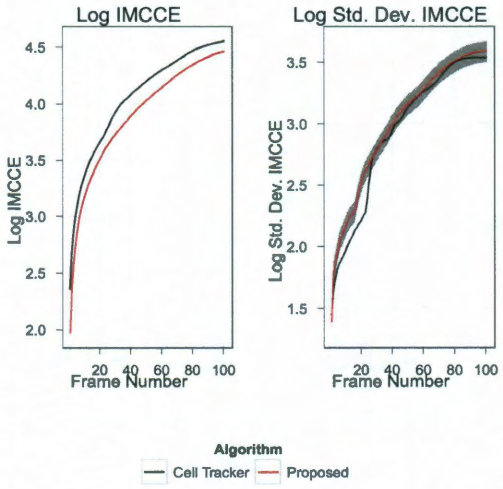


Figure 5.11 : Integrated Mean Cytoplasmic Classification Error for simulation 411.

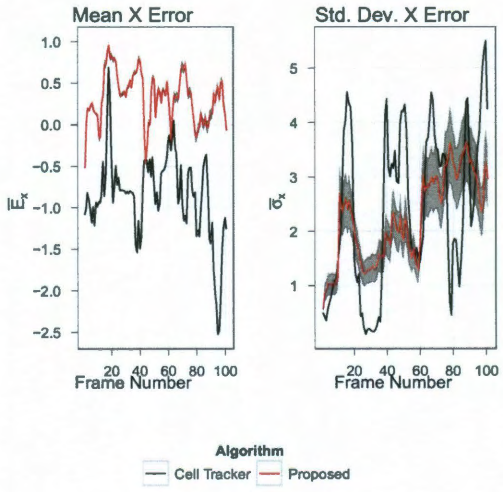


Figure 5.12 : Average signed error in X coordinate of nuclear ellipse for each frame in simulation 411.

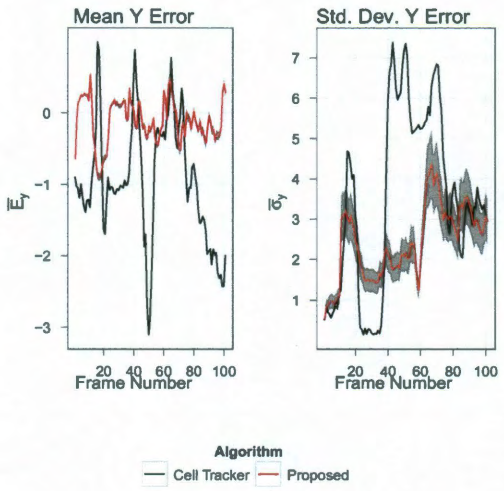


Figure 5.13 : Average signed error in Y coordinate of nuclear ellipse for each frame in simulation 411.

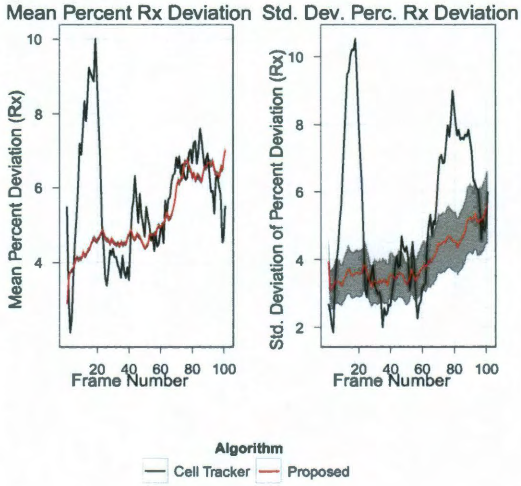


Figure 5.14 : Average percent deviation in major axis half-length for nuclear ellipses for each frame in simulation 411.

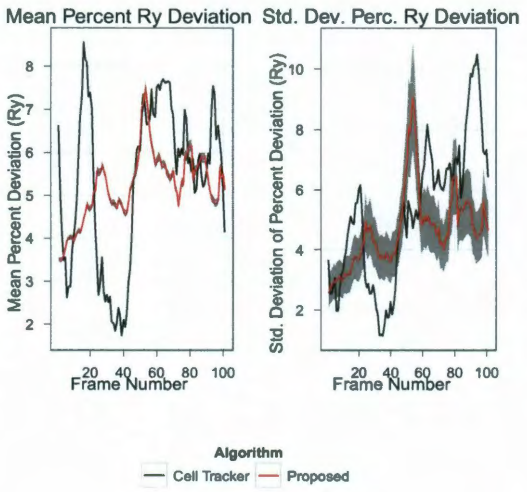


Figure 5.15 : Average percent deviation in minor axis half-length for nuclear ellipses for each frame in simulation 411.

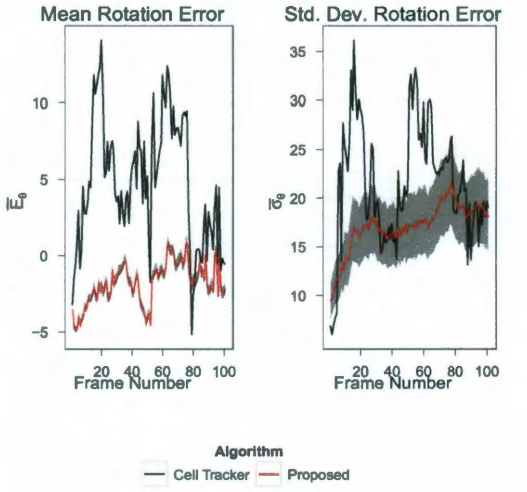


Figure 5.16 : Average signed error in rotation of major axis for nuclear ellipse for each frame in simulation 411.

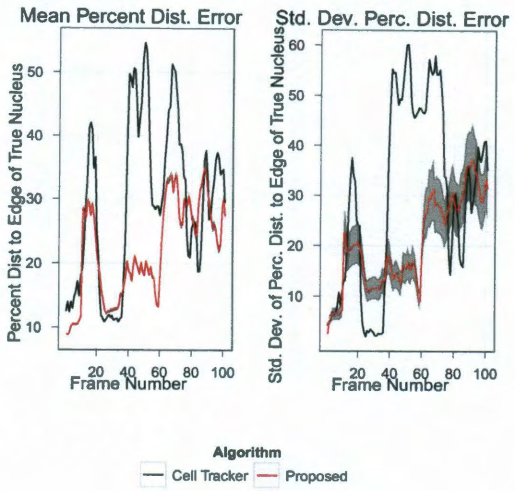


Figure 5.17 : Average distance error between true and estimated nuclear centers as a percentage of the nuclear radius in the direction of error. (simulation 411)

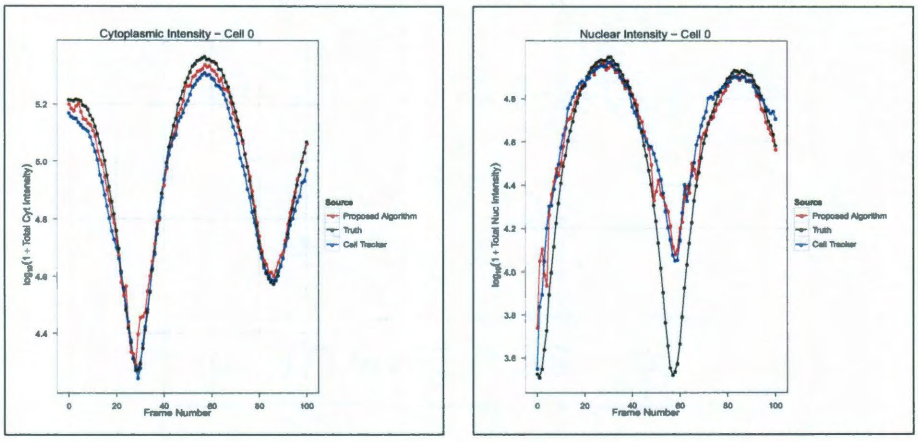


Figure 5.18 : Total intensity levels calculated for cell 0 of simulation 411.

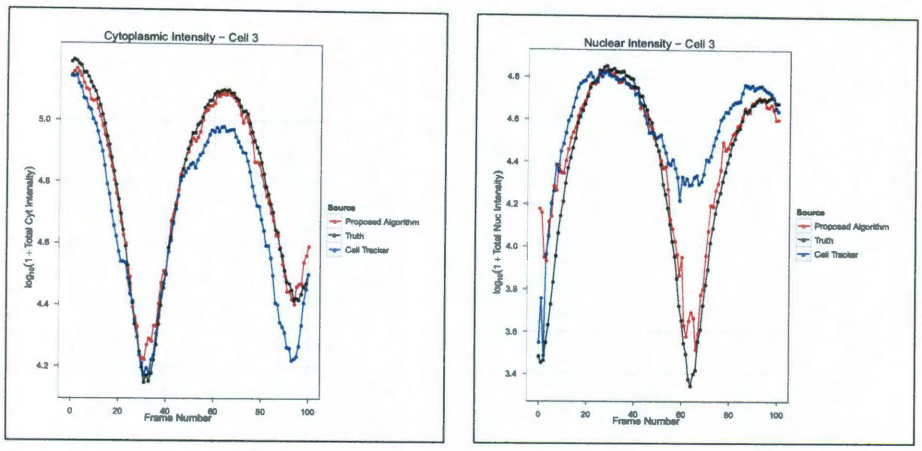


Figure 5.19 : Total intensity levels calculated for cell 3 of simulation 411.

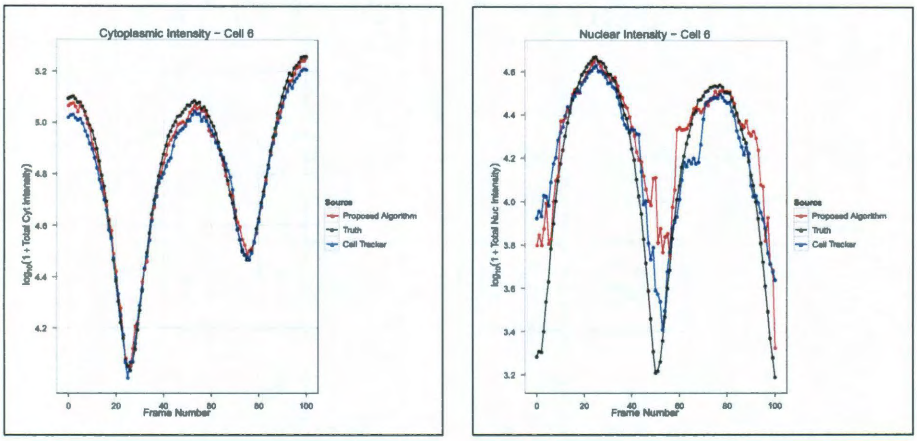


Figure 5.20 : Total intensity levels calculated for cell 6 of simulation 411.

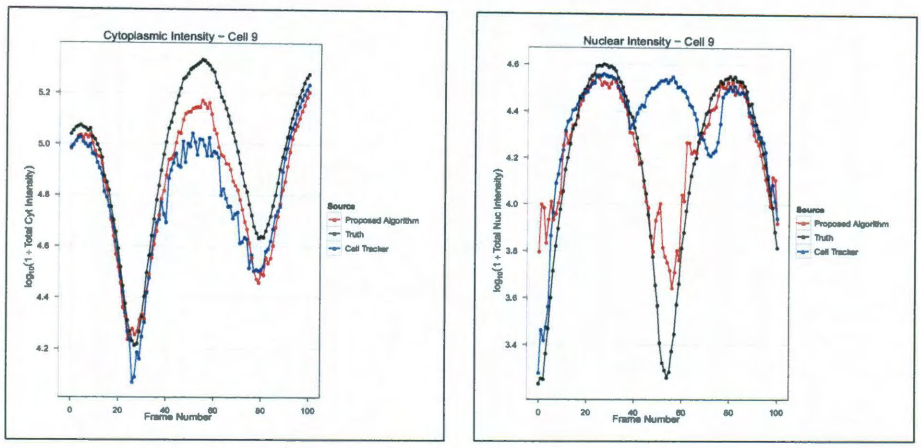


Figure 5.21 : Total intensity levels calculated for cell 9 of simulation 411.

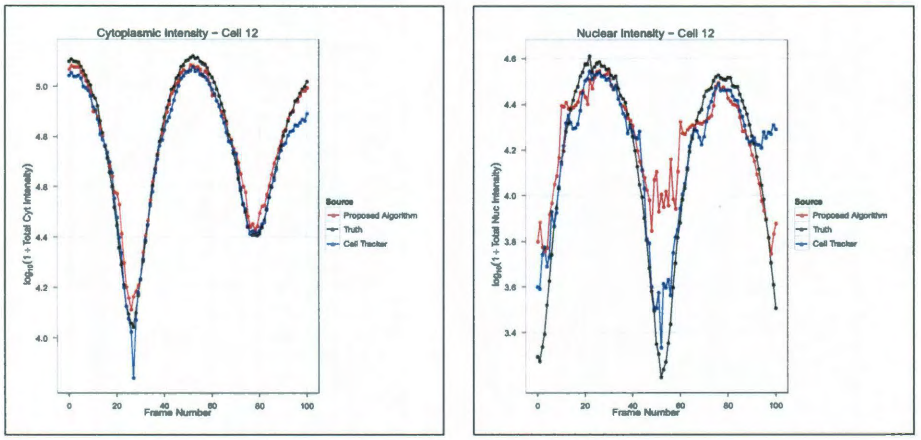


Figure 5.22 : Total intensity levels calculated for cell 12 of simulation 411.

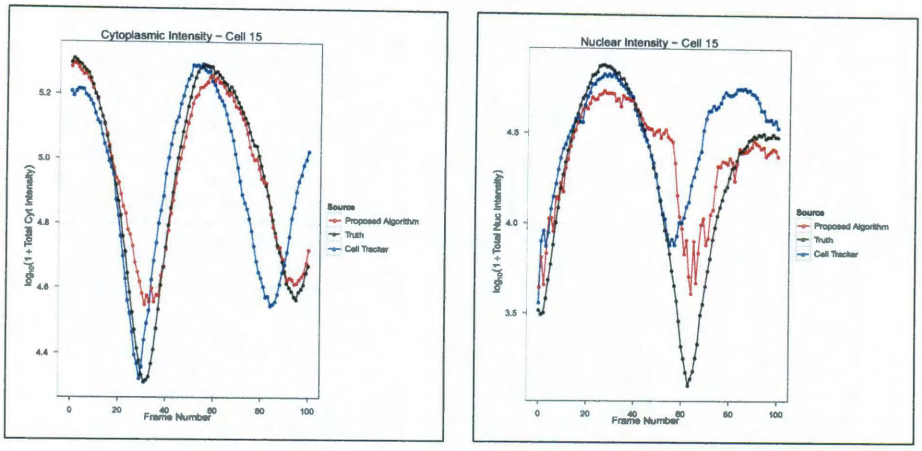


Figure 5.23 : Total intensity levels calculated for cell 15 of simulation 411.

Medium Maximum Percent Distance Error

Simulation 207 was generated using a dense collection of large cells with medium nuclei. The cells are generated from regular polygons with 25 corners with fast movement and medium translocation assumed. The simulation successfully updated 1,150 times before failure. Select frames from this simulation are shown in Figure 5.24.

Figure 5.25 indicates a lack of any significant change in the cellular positions during this simulation. The nuclei shift as expected from the simulation parameters. It is likely that reducing or removing the rest period at the end of the movement cycle would cause an obvious shift in cellular positions.

The proposed algorithm was able to correctly detect three of the five cells in the image, where Cell Tracker was only able to detect two with an extraneous nucleus. Although the cell in the bottom left corner of the screen is tracked successfully in

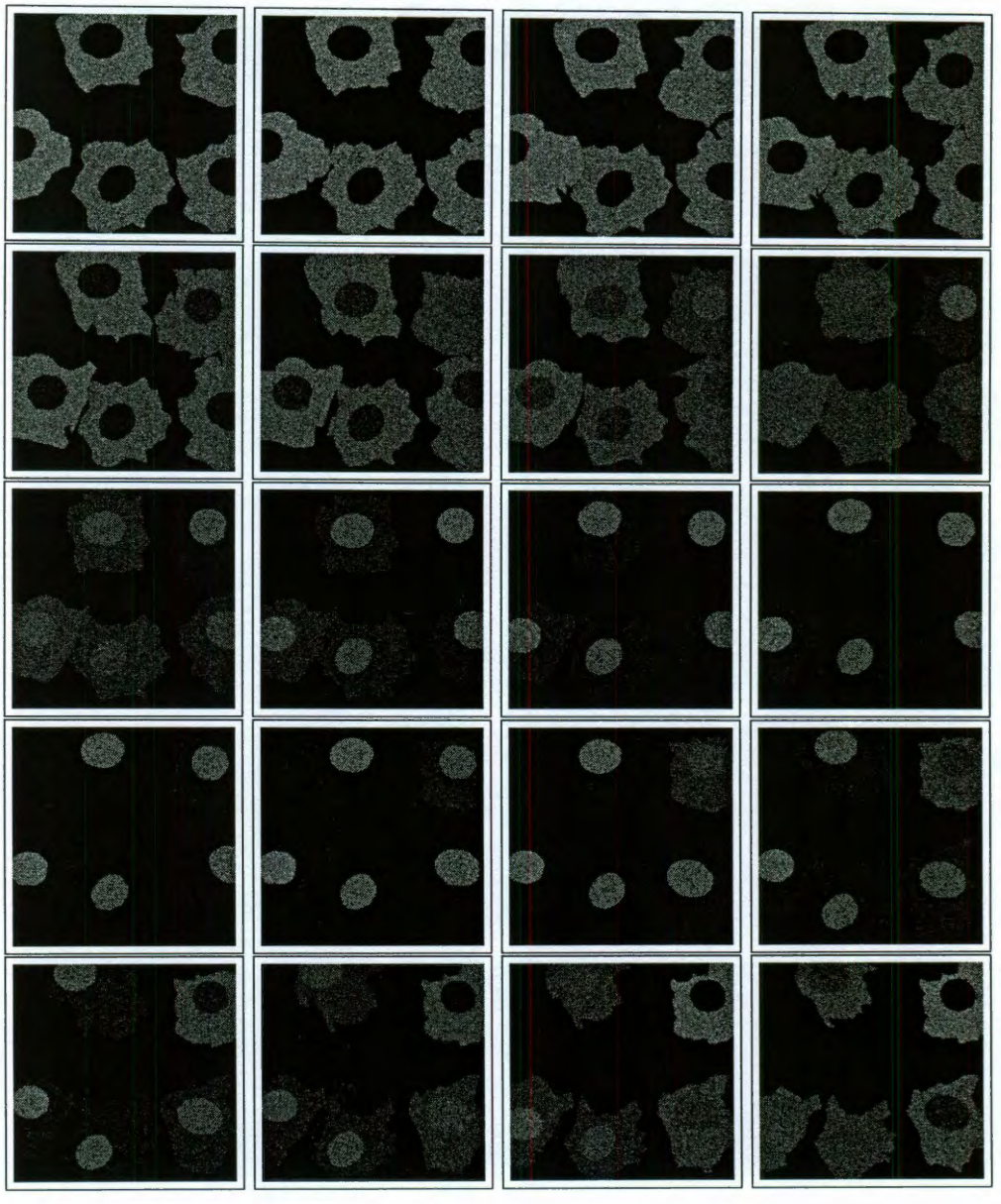


Figure 5.24 : Raw data from simulation 207 (Parameter details in text)

the cytoplasmic component, the proposed algorithm was unable able to shift the size of the nuclear ellipse to sufficiently cover the entire area. The tracking failure for this nucleus is due to the limited amount of resizing that is allowed by the proposed algorithm from one frame to the next.

This simulation also showcases another of Cell Tracker's major flaws. Although Cell Tracker does as well in tracking the true nuclei as the proposed algorithm, Cell Tracker's cytoplasmic boundary routine contains major flaws. As can be seen in Figure 5.26, when tracked and untracked cells come in contact, the boundaries from the tracked cell are extended to include the untracked cell. Because the used measurement is the integrated intensity in each region, this type of error can be catastrophic. A scientist using Cell Tracker would have to manually redraw the boundaries for every frame in which the boundaries are not correctly delimited – an option which consumes a large amount of time and manpower.

Figures 5.27-5.35 show the error metrics for Cell Tracker and the mean error metrics for 100 tracking runs for this data, as well as the 95% confidence interval for the mean. Figures 5.28-5.35 also show the average standard deviation between cells in each frame and the 95% confidence interval for this average.

Figure 5.35 shows the percent error distance from the proposed algorithm does show a significant upswing near the end of this tracking run. This upswing is due to two different factors. The first is the loss of the nucleus of the top cell near the end of the tracking run. One possible cause is the nucleus moving at a speed which

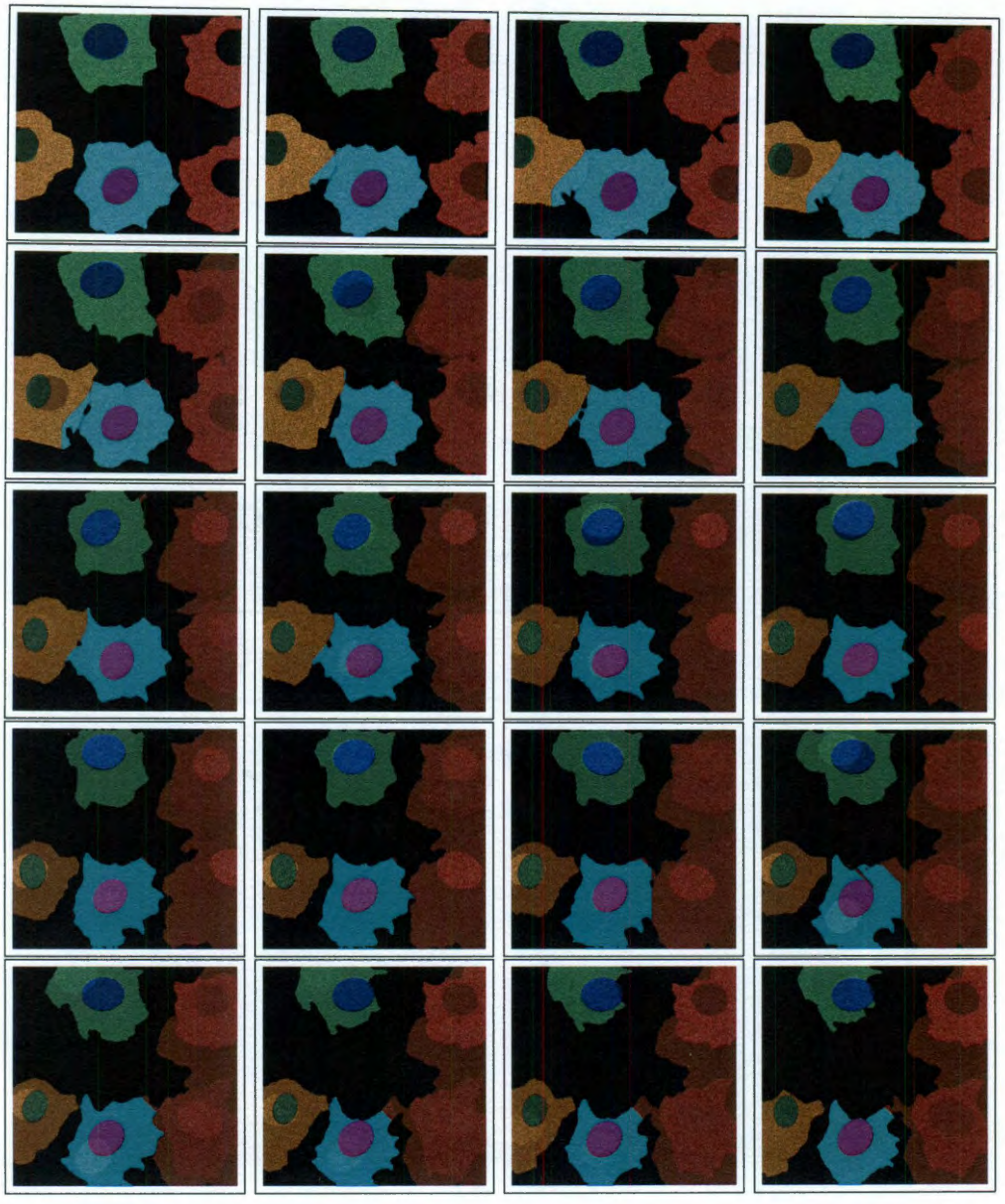


Figure 5.25 : Visualized tracking results from simulation 207 using the proposed algorithm. The regions are overlaid on the raw data from Figure 5.24.

Figure 5.26 : Visualized tracking results from simulation 207 using Cell Tracker. The region boundaries are overlaid on the raw data from Figure 5.24.

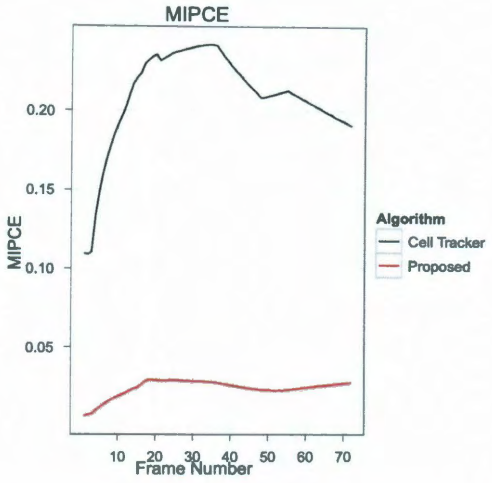


Figure 5.27 : Mean Integrated Percent Classification Error for simulation 207.

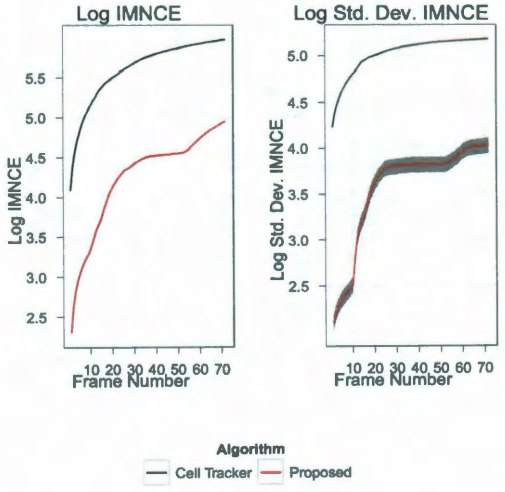


Figure 5.28 : Integrated Mean Nuclear Classification Error for simulation 207.

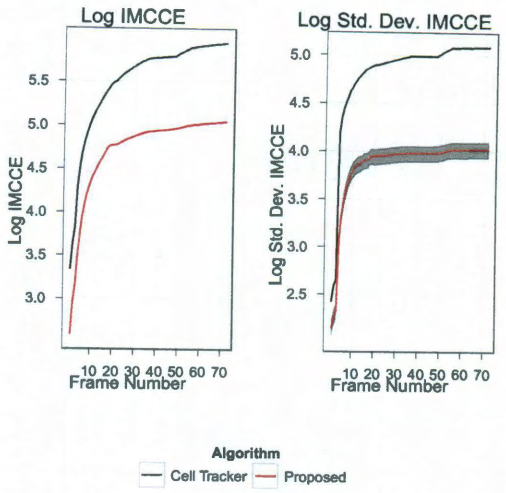


Figure 5.29 : Integrated Mean Cytoplasmic Classification Error for simulation 207.

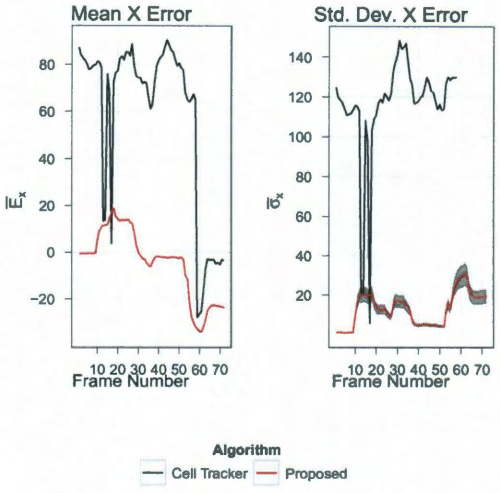


Figure 5.30 : Average signed error in X coordinate of nuclear ellipse for each frame in simulation 207.

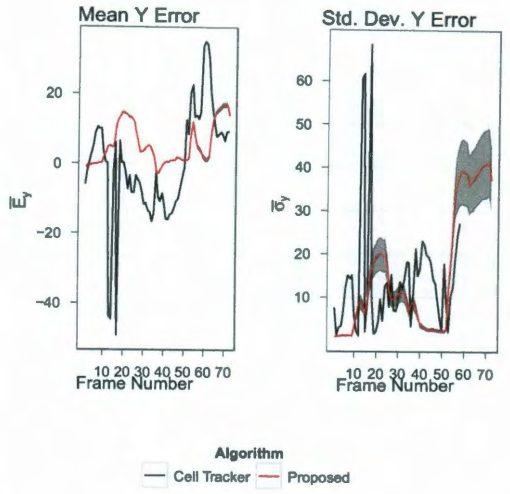


Figure 5.31 : Average signed error in Y coordinate of nuclear ellipse for each frame in simulation 207.

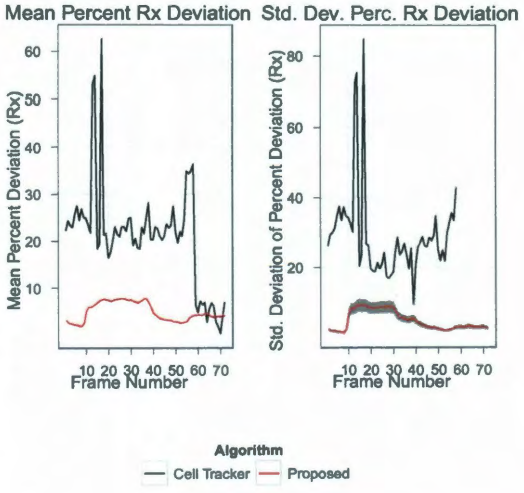


Figure 5.32 : Average percent deviation in major axis half-length for nuclear ellipses for each frame in simulation 207.

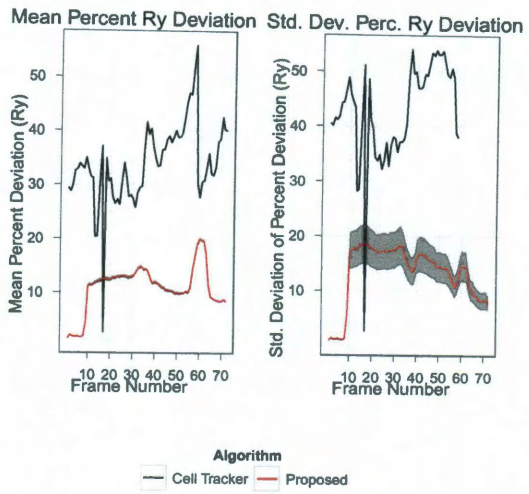


Figure 5.33 : Average percent deviation in minor axis half-length for nuclear ellipses for each frame in simulation 207.

is too rapid for the proposed algorithm to match with the parameters used. Because the edges of the nuclei during these end points can clearly be seen and because Cell Tracker suffers from the same error at the same time, a reasonable conclusion is that the algorithm as parameterized simply cannot accurately track the rapid shifts in nuclear position. The second factor that causes the upswing in error at the end of the run is the loss of one of the nuclei from the screen. Figure 5.24 shows the nucleus for the cell in the lower left corner of the image leaving the screen near the end of the simulation. The proposed algorithm is unable to account for nuclei leaving the screen, resulting in a percentage error based on incorrectly small or undefined results.

Figures 5.36-5.38 show the accuracy of the proposed algorithm when calculating

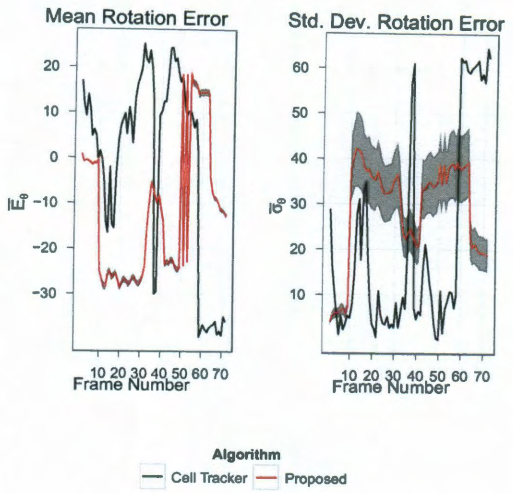


Figure 5.34 : Average signed error in rotation of major axis for nuclear ellipse for each frame in simulation 207.

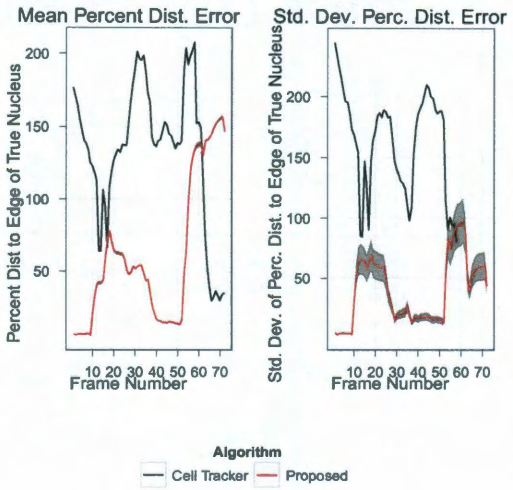


Figure 5.35 : Average distance error between true and estimated nuclear centers as a percentage of the nuclear radius in the direction of error. (simulation 207)

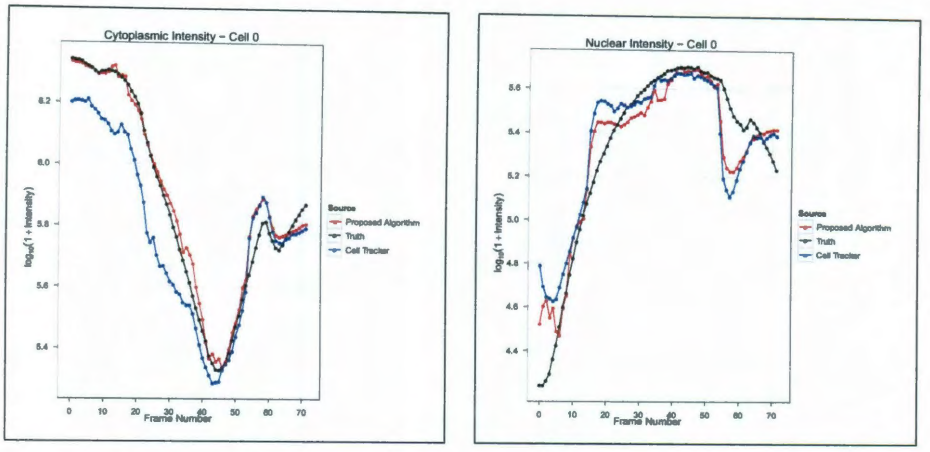


Figure 5.36 : Total intensity levels calculated for cell 0 of simulation 207.

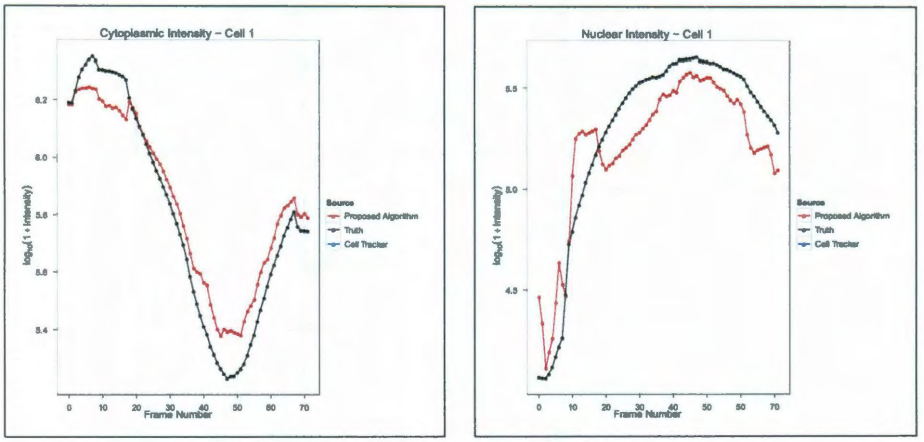


Figure 5.37 : Total intensity levels calculated for cell 1 of simulation 207.

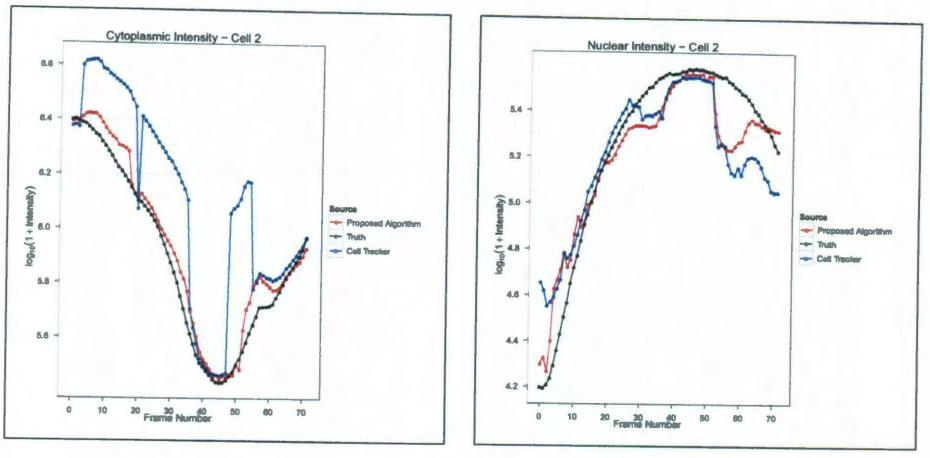


Figure 5.38 : Total intensity levels calculated for cell 2 of simulation 207.

the integrated region intensity for the nuclei and cytoplasmic areas of each cell. There is no Cell Tracker for Figure 5.37 because it is the cell that Cell Tracker was unable to segment initially. It can be seen that the proposed algorithm is more accurate than Cell Tracker for the cells that are tracked. Although the visual tracking results were off at times, the calculated region intensities are good approximations of the true values.

High Maximum Percent Distance Error

Simulation 477 was generated using a medium collection of medium cells with medium nuclei. The cells are generated from regular polygons with 10 corners. These cells are assumed to undergo medium movement and medium speed translocation. The simulation successfully completed 1,675 iterations before failure. Select frames from

this simulation are shown in Figure 5.39.

Figure 5.40 shown visible evidence of cellular movement in this image series. Once again, the proposed algorithm is unable to detect several cells in the image, leading to only four of the potential nine cells actually being tracked. However, the same is true for Cell Tracker's segmentation algorithm. The other five cells have too much of the nuclei outside of the image to be detected in the initial frame.

Visually, the proposed algorithm does a better job of Cell Tracker in tracking these cells. This is most evident which following the track of the nucleus in the upper left corner of the image. The last line of Figure 5.41 shows the nuclear boundary being pulled out to the cytoplasmic edge. The last line of Figure 5.40 shows that the proposed algorithm correctly tracks this ellipse. In addition, Cell Tracker expands the cytoplasmic boundaries into untracked cells in several frames in this simulation. Figure 5.40 shows that the proposed algorithm avoids this problem so long as the two cells are not touching in the initial frame.

Figures 5.42-5.50 show the error metrics for Cell Tracker and the mean error metrics for 100 tracking runs for this data, as well as the 95% confidence interval for the mean. The error metrics for the proposed algorithm indicate the values tracking the shape of the error produced by Cell Tracker, although lower.

Figures 5.51-5.54 show the proposed algorithm calculating the integrated region intensity more accurately than Cell Tracker. This is to be expected, because the error metrics on the proposed algorithm also indicate a better tracking result.

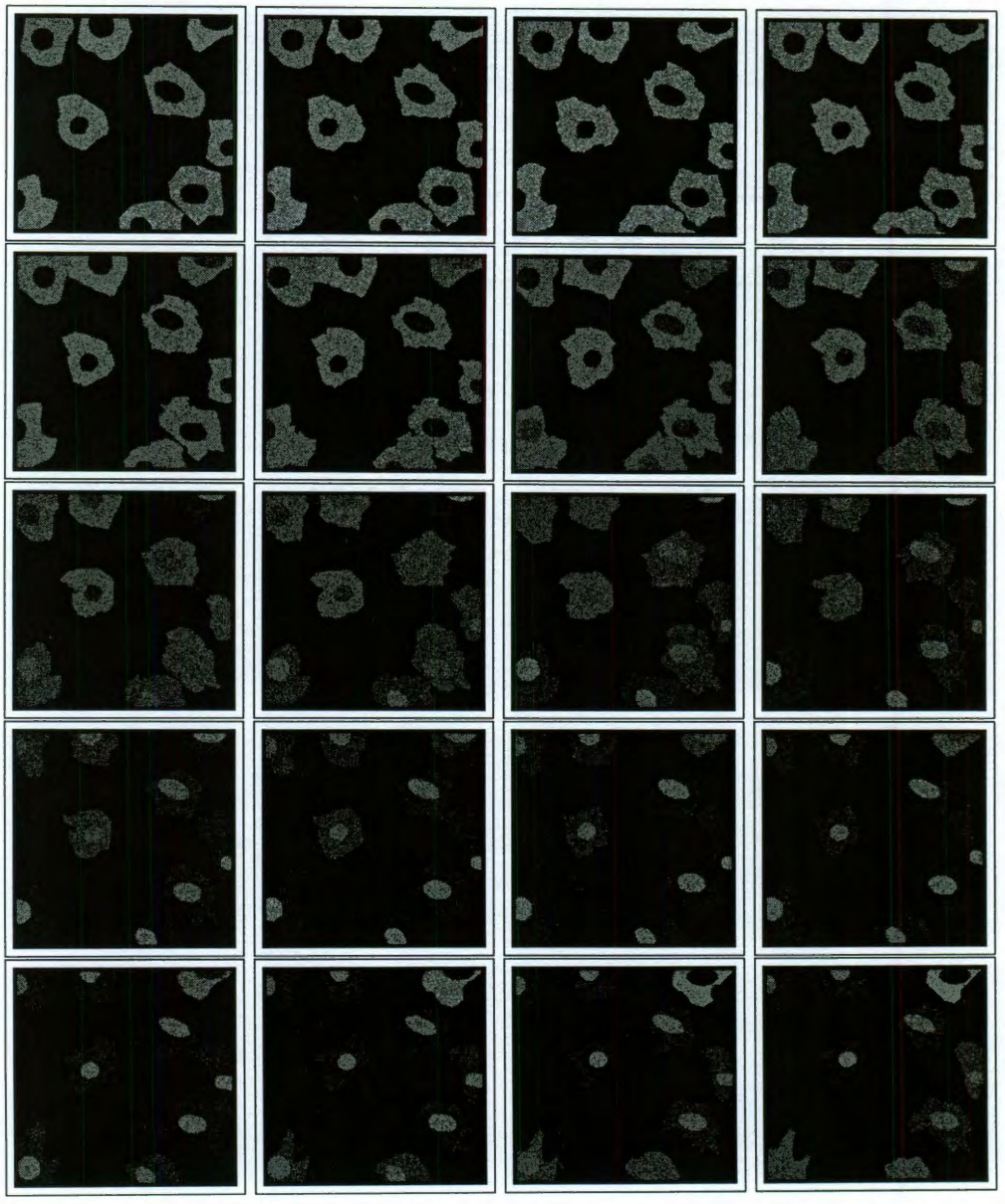


Figure 5.39 : Raw data from simulation 477 (Parameter details in text)

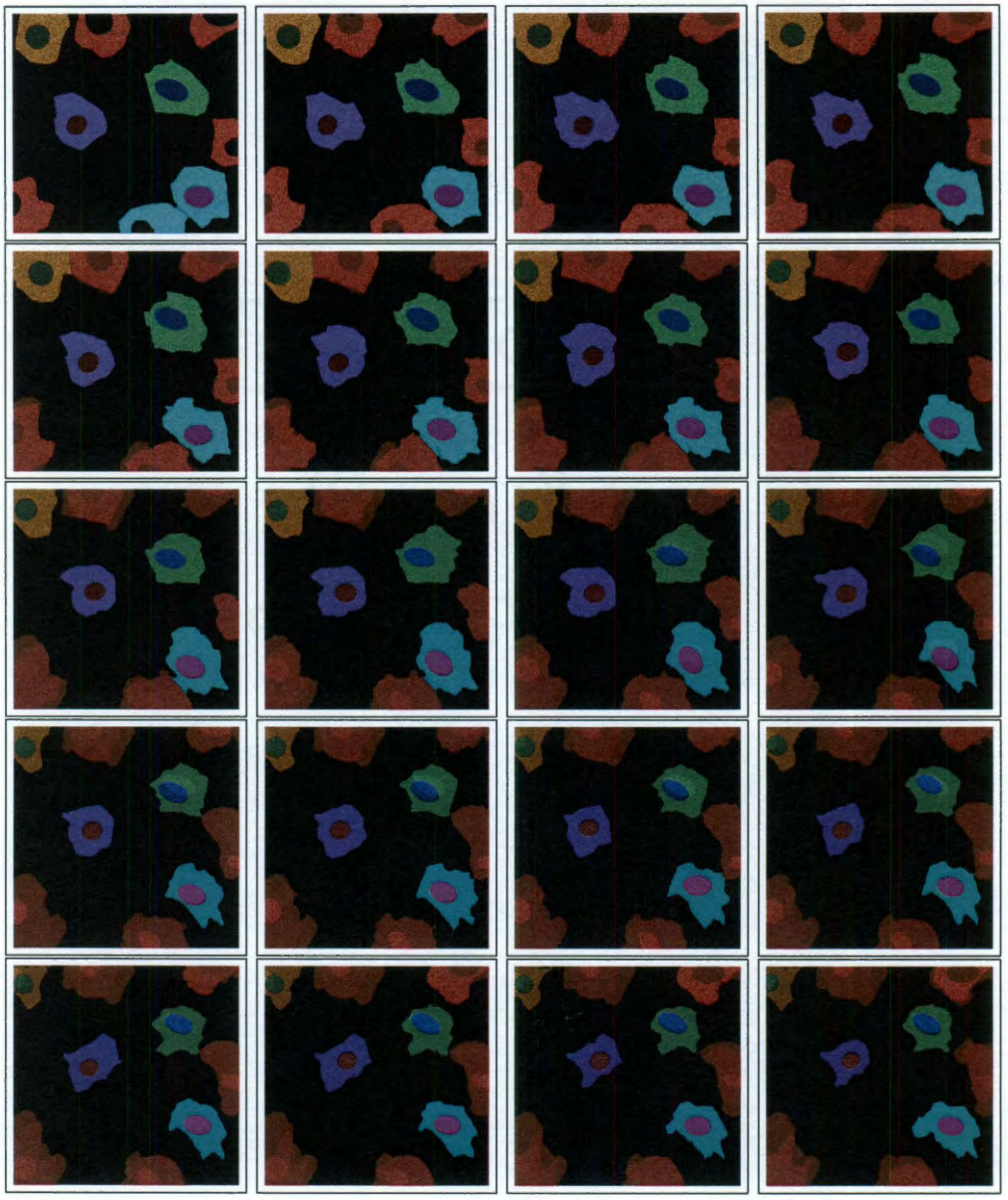


Figure 5.40 : Visualized tracking results from simulation 477 using the proposed algorithm. The regions are overlaid on the raw data from Figure 5.39.

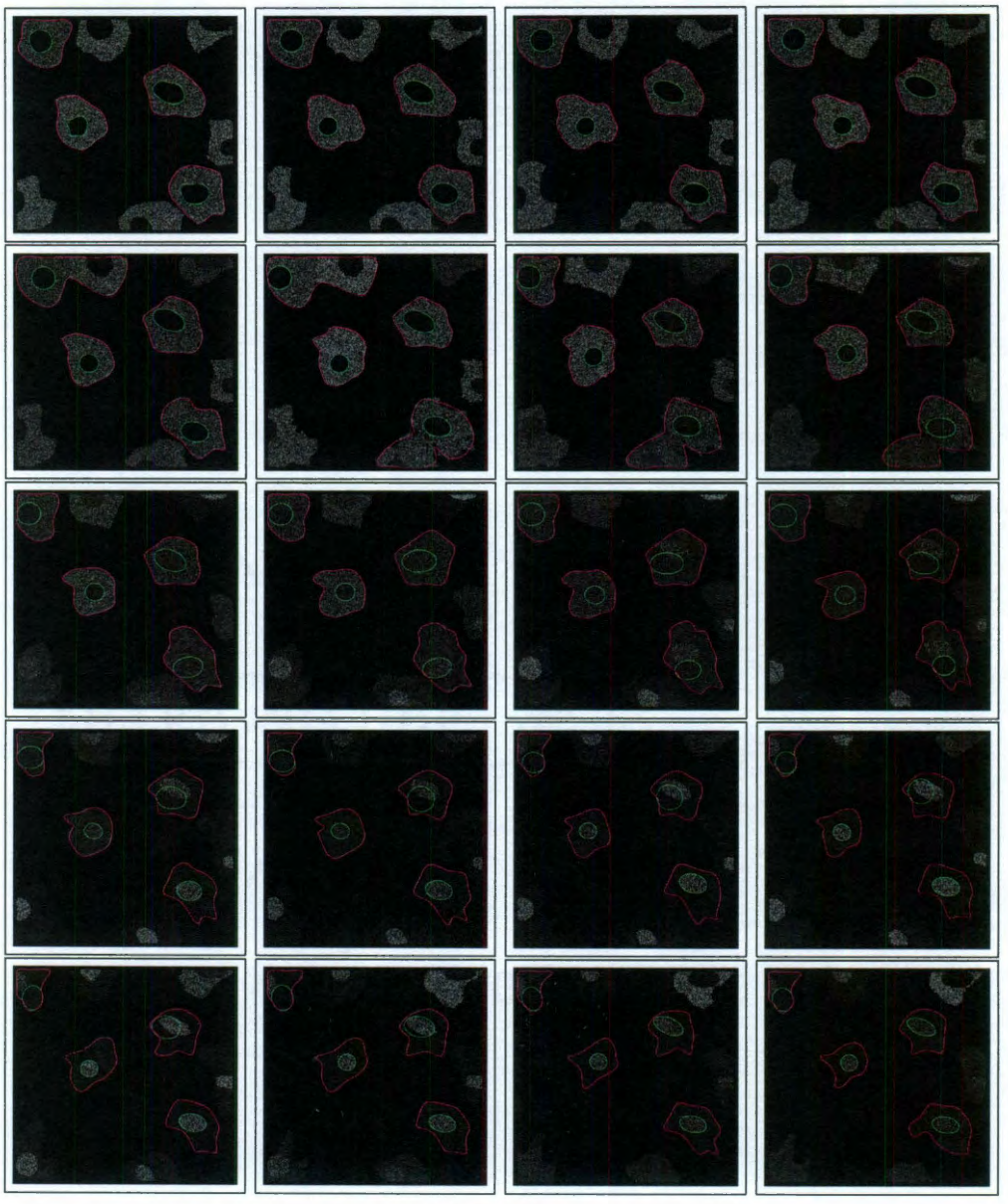


Figure 5.41 : Visualized tracking results from simulation 477 using Cell Tracker. The region boundaries are overlaid on the raw data from Figure 5.39.

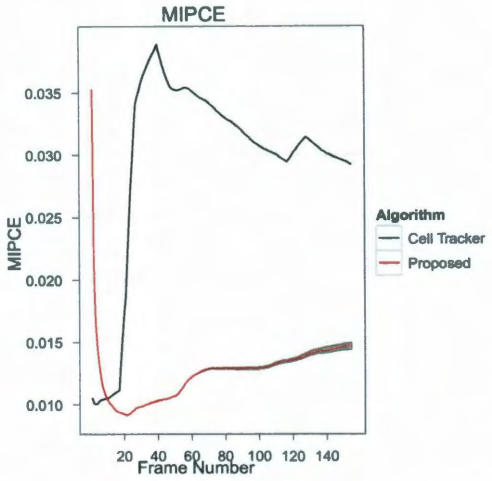


Figure 5.42 : Mean Integrated Percent Classification Error for simulation 477.

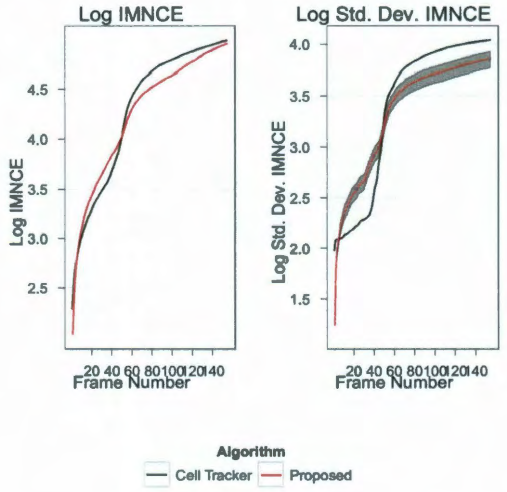


Figure 5.43 : Integrated Mean Nuclear Classification Error for simulation 477.

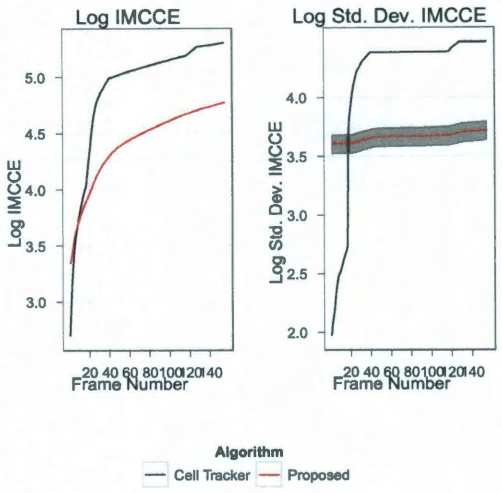


Figure 5.44 : Integrated Mean Cytoplasmic Classification Error for simulation 477.

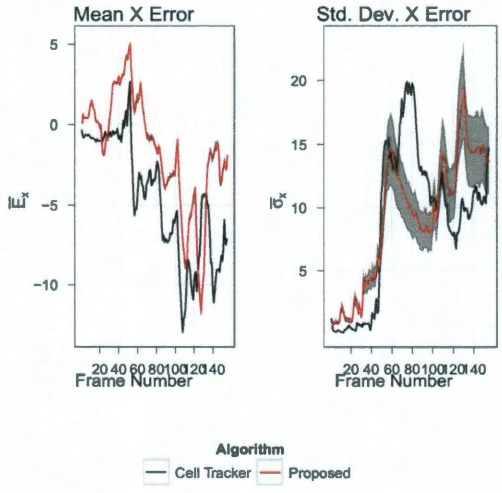


Figure 5.45 : Average signed error in X coordinate of nuclear ellipse for each frame in simulation 477.

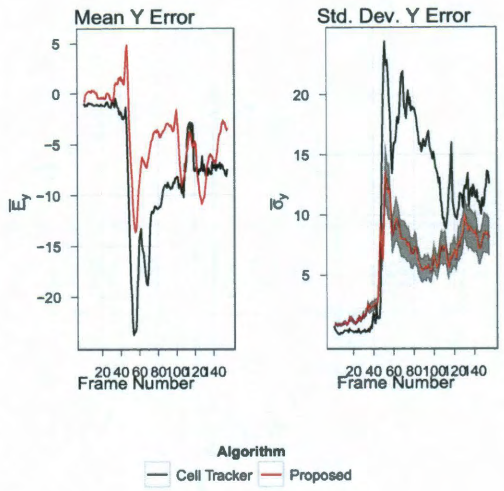


Figure 5.46 : Average signed error in Y coordinate of nuclear ellipse for each frame in simulation 477.

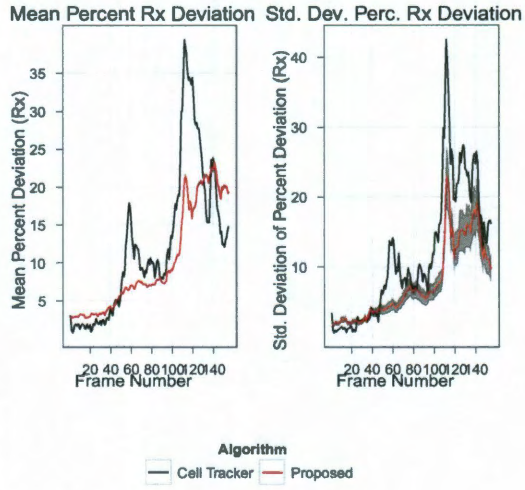


Figure 5.47 : Average percent deviation in major axis half-length for nuclear ellipses for each frame in simulation 477.

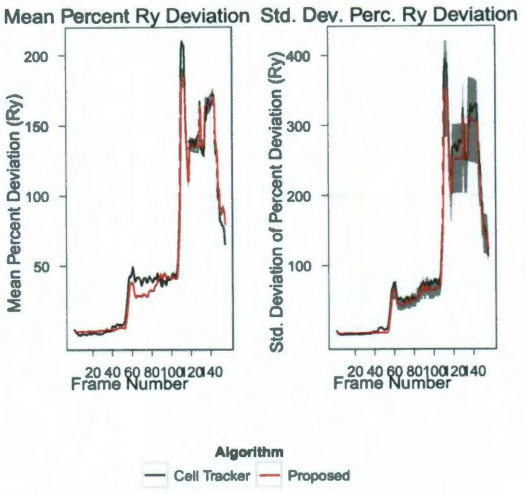


Figure 5.48 : Average percent deviation in minor axis half-length for nuclear ellipses for each frame in simulation 477.

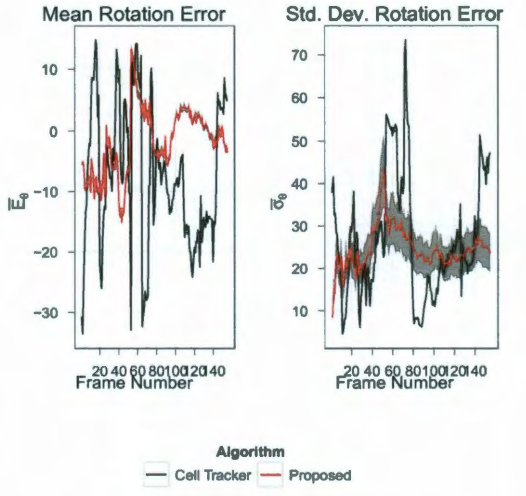


Figure 5.49 : Average signed error in rotation of major axis for nuclear ellipse for each frame in simulation 477.

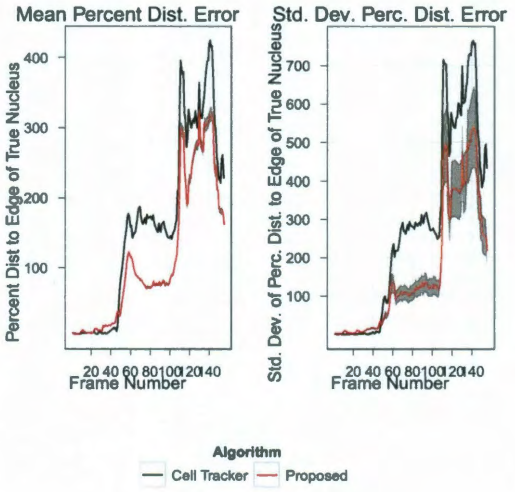


Figure 5.50 : Average distance error between true and estimated nuclear centers as a percentage of the nuclear radius in the direction of error. (simulation 477)

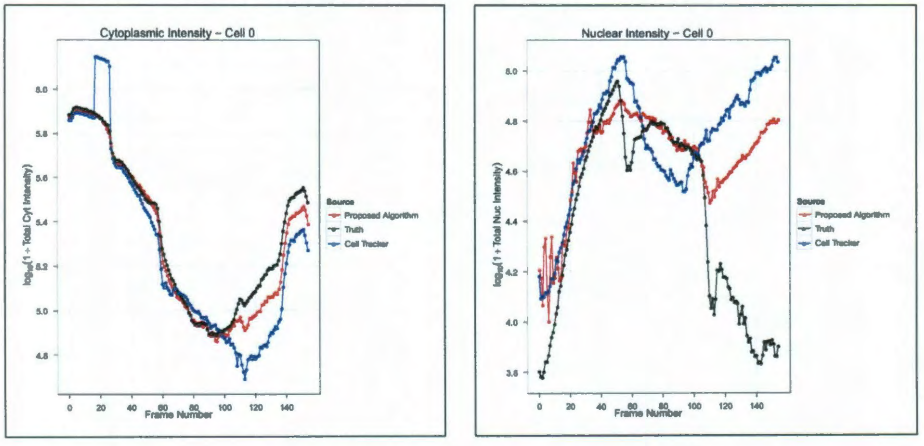


Figure 5.51 : Total intensity levels calculated for cell 0 of simulation 477.

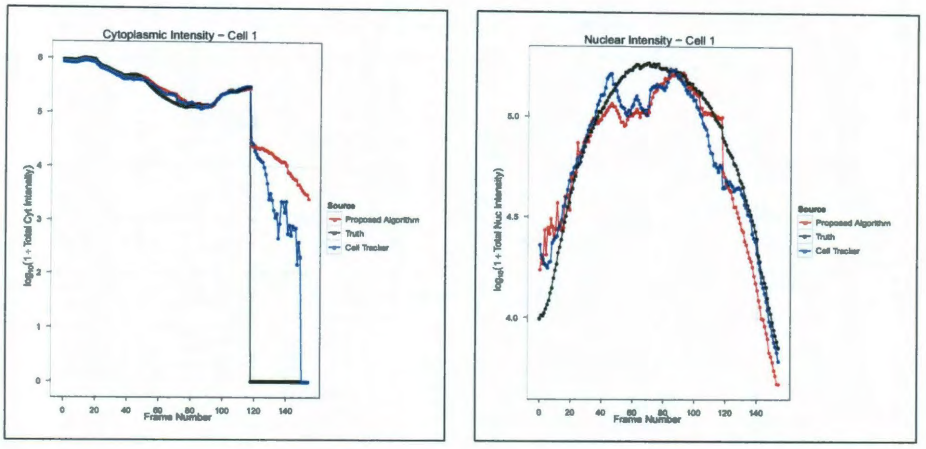


Figure 5.52 : Total intensity levels calculated for cell 1 of simulation 477.

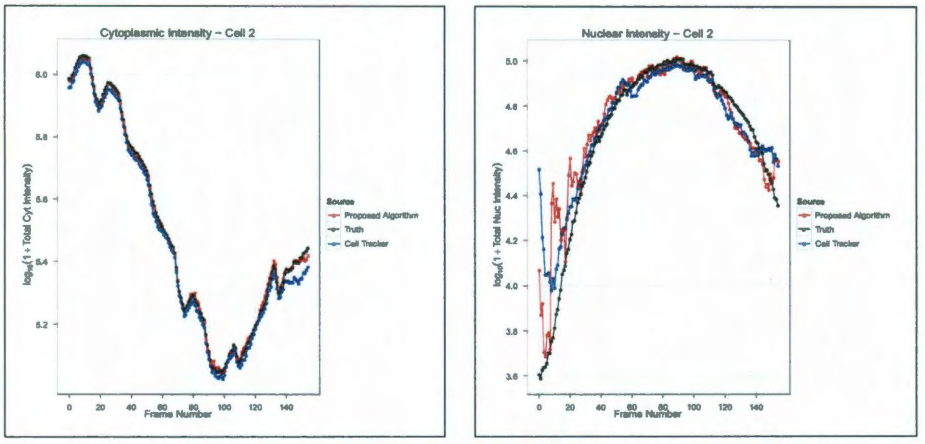


Figure 5.53 : Total intensity levels calculated for cell 2 of simulation 477.

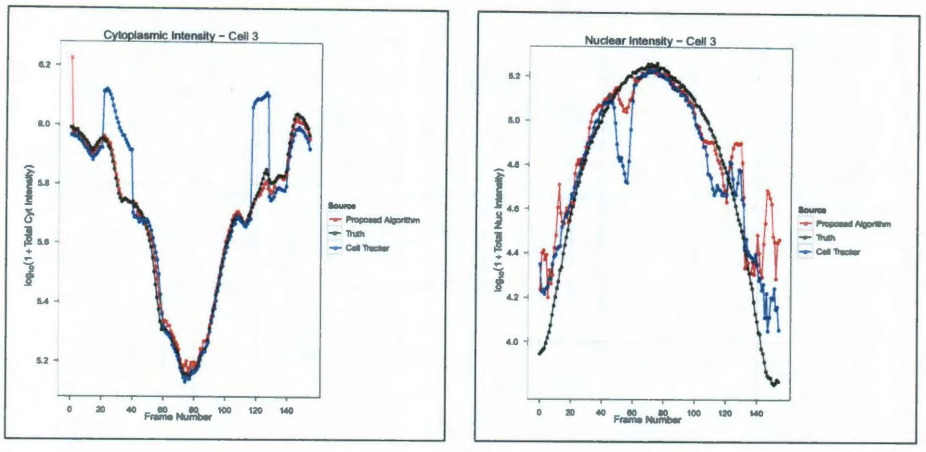


Figure 5.54 : Total intensity levels calculated for cell 3 of simulation 477.

5.2.2 Experimental Data Tracking

This section describes the visual comparison between tracking results from the proposed algorithm for experimental data and the results obtained by Cell Tracker for the same data. The visual comparison will be analyzed for three sets of experimental data. The first data set was provided as a demo for the Cell Tracker software. The other two data sets were provided by Dr. Allan Brasier from the University of Texas Medical Branch in Galveston, Texas.

Experimental Data Set 1

This is the base experimental data set used to test the initial segmentation in Chapter 3.3. This data set has been split into three channels, with a single channel to delineate the cytoplasm boundaries in the image. As the proposed algorithm is designed to only

use a single fluorescence channel, the the cytoplasmic channel will be ignored. The discussion will concentrate solely on the protein translocation channel. A selection of frames from this first data set are shown in Figure 5.55.

This data set showcases the differences between the initial segmentation methods. As can clearly be see in Figure 5.57, Cell Tracker fails to recognize nuclei which are touching the cytoplasmic membrane. The proposed segmentation method accomplishes this task, as can be seen in Figure 5.56. However, the proposed method fails to segment the nuclear membrane on the cell in the bottom middle of the image. As a result, that entire cell remains untracked.

In the results from Cell Tracker, it can be seen in Figure 5.57 that the nucleus that stays bright is consistently and correctly tracked, although the segmentation of the associated cytoplasm appears random at best. In addition, the nucleus of the isolated cell is drawn off to the edges of the cytoplasm as in our simulated examples.

The proposed algorithm is successful at keeping the three clustered cells separate. Although the proposed algorithm is not as accurate in tracking the nuclear boundary of the consistently bright nucleus, this is due to the presence of edges resulting from the nucleoli in the image. The nucleoli project constant edges in the image, which draw the ellipse away from the true nuclear edges. Although we manage to "lose" the nucleus of the blue cell around halfway through the tracking run, the other two nuclei match the nuclear boundaries very well.

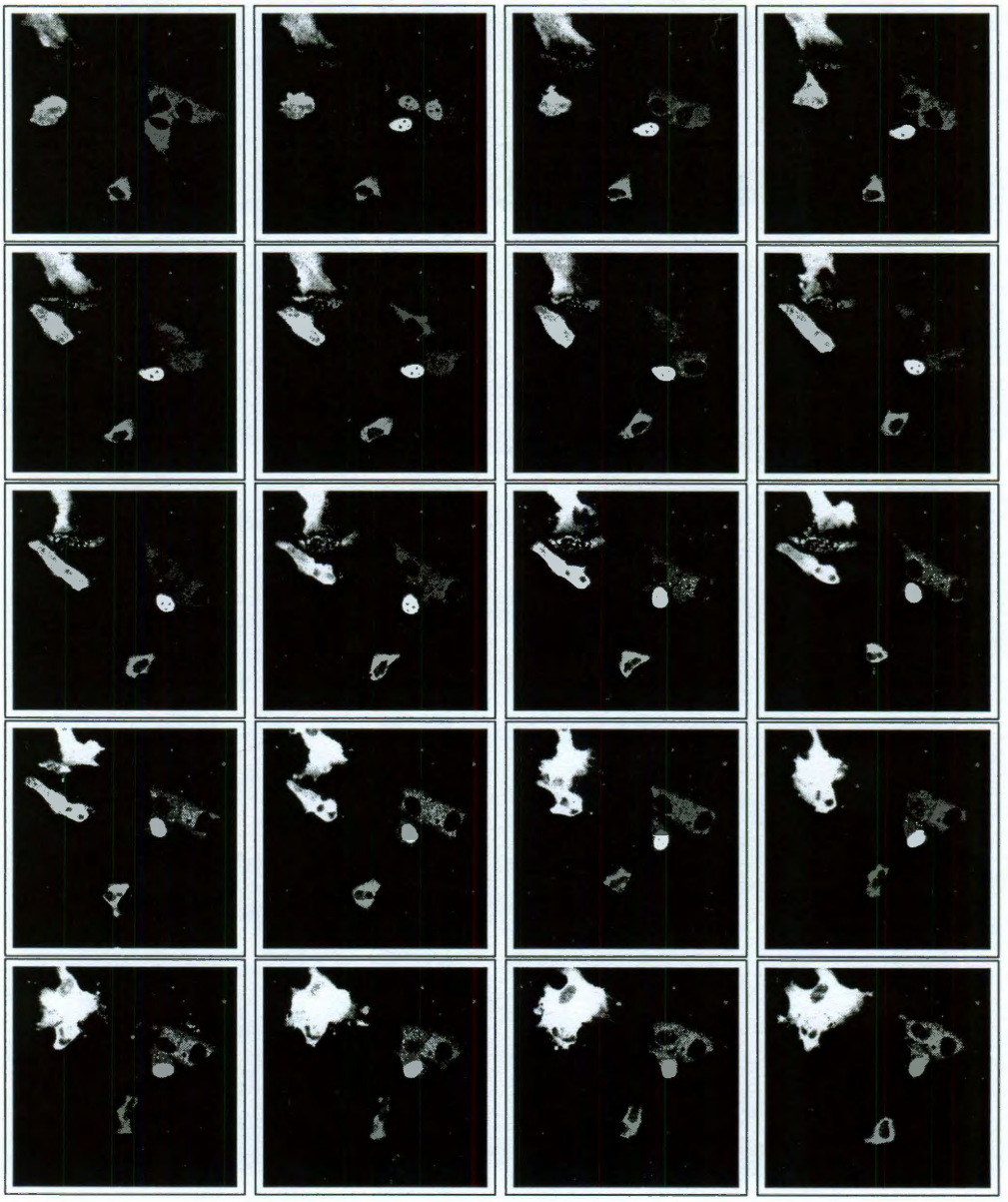


Figure 5.55 : Raw data from the first experimental time series. Data provided with Cell Tracker software.

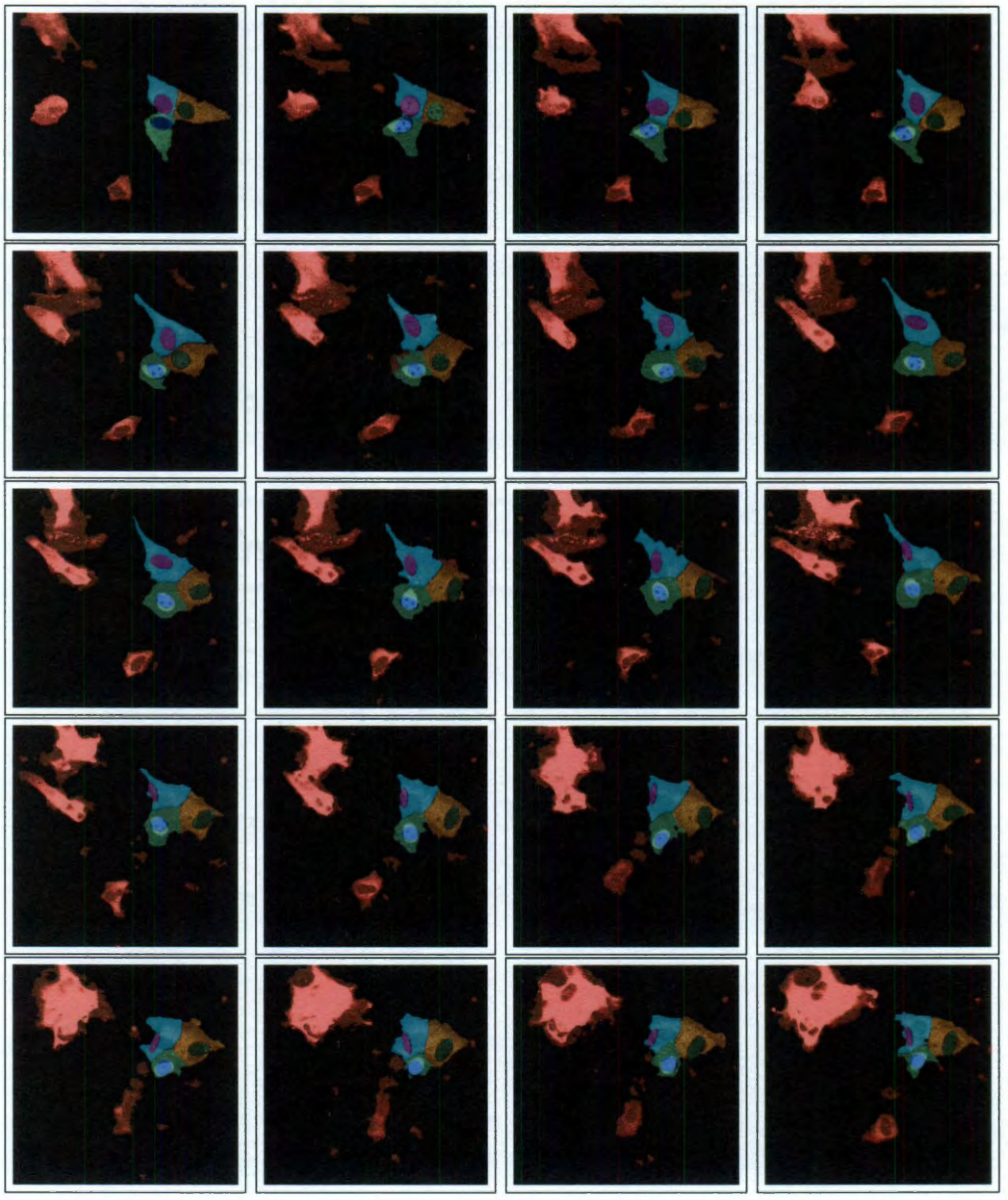


Figure 5.56 : Visualized tracking results from experimental data set 1 using the proposed algorithm. The regions are overlaid on the raw data from Figure 5.55.

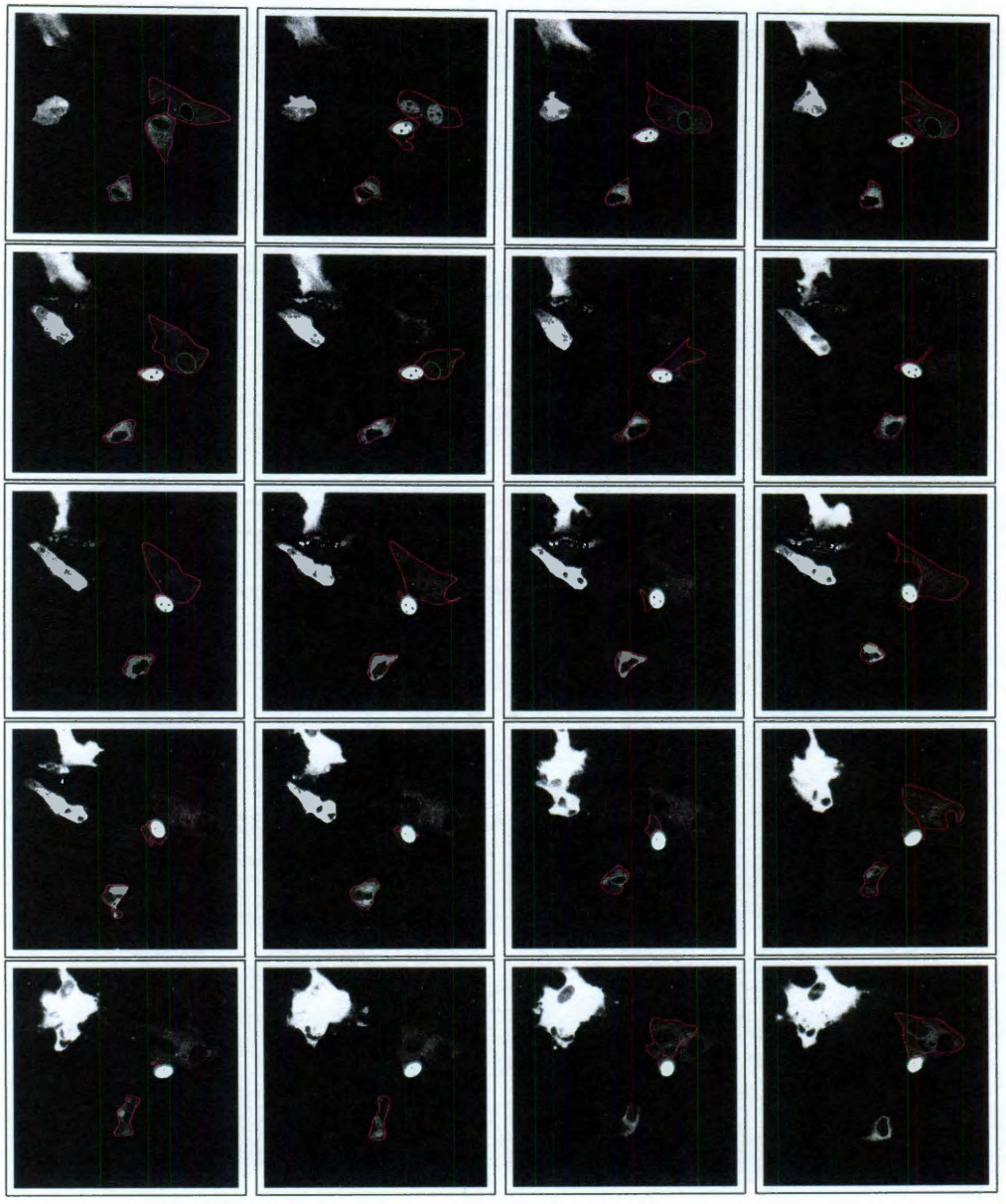


Figure 5.57 : Visualized tracking results from experimental data set 1 using Cell Tracker. The region boundaries are overlaid on the raw data from Figure 5.55.

Experimental Data Set 2

The second experimental data set was produced by Dr. Brasier's group in Galveston. This data set poses several challenges to the tracking algorithm, including a large variation in the overall brightness of the cells, as well as having two nuclei very close together in the center top of the original image. Select frames from this data set are shown in Figure 5.58.

The initial segmentation in the first frame of Figure 5.59 shows that the proposed algorithm is able to determine the nuclear and cytoplasm boundaries for all the cells in this image. We do note, however, that the nucleus of the green cell does not correctly match the nuclear area. We will see that this nucleus never quite manages to correctly estimate the true nucleus of the cell. Another problem is the extension of the light blue cell in the bottom left corner of the first image into another cell that is partially off the screen.

The first frame of Figure 5.60 shows the segmentation results for Cell Tracker, which is worse than the initial segmentation of the proposed algorithm. The primary difference between the two algorithms is the inability of Cell Tracker to recognize the nucleus of the lower middle cell.

In tracking the cells, Cell Tracker has only mediocre results. Cell Tracker quickly loses the the cell in the lower left corner of the screen. Also, Cell Tracker once again merges the close nuclei – a very serious error. In comparison to Cell Tracker, the proposed algorithm manages to correctly track the cell in the upper left corner of

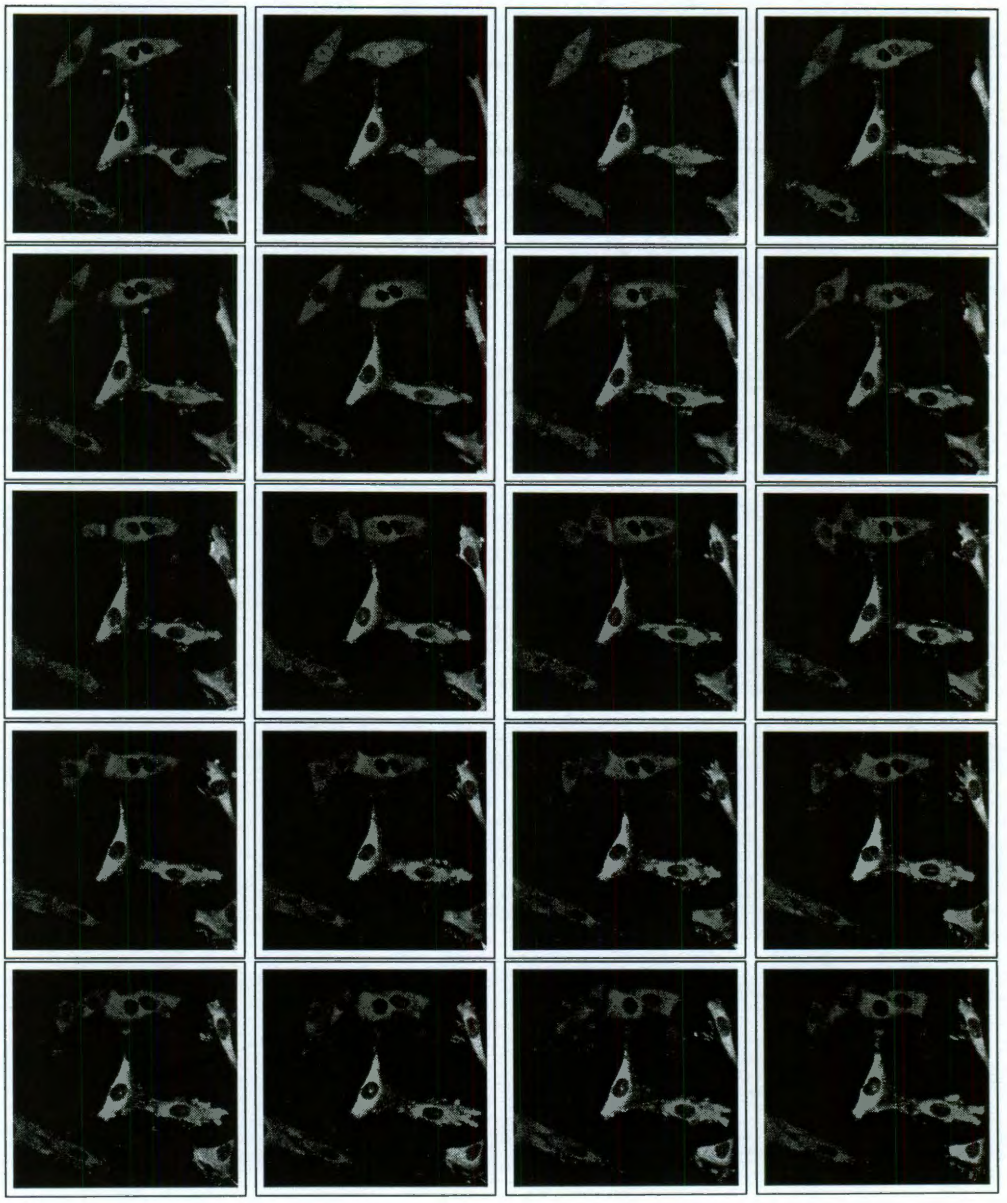


Figure 5.58 : Raw data from the second experimental time series. Data provided by Dr. Allan Brasier.

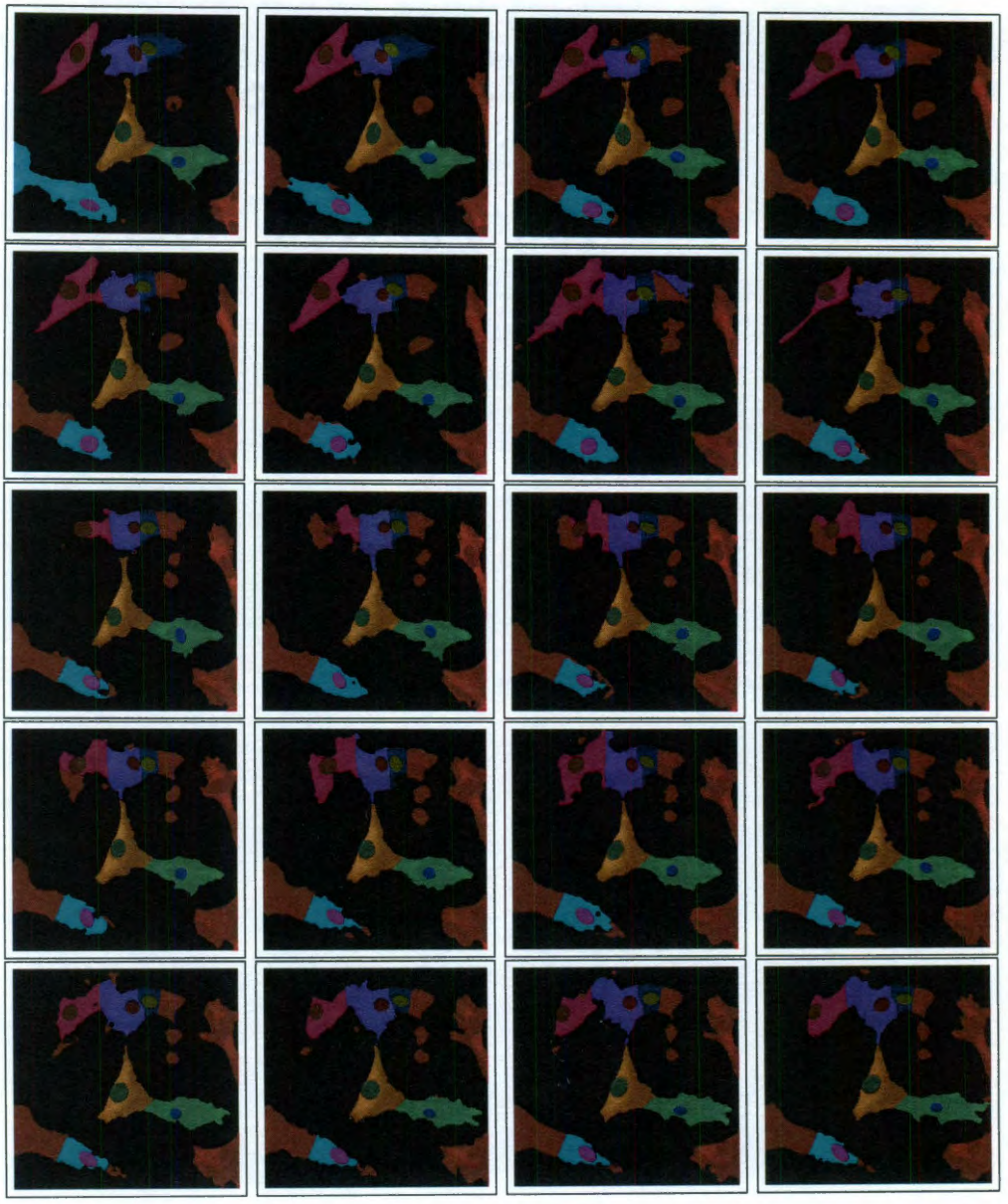


Figure 5.59 : Visualized tracking results from experimental data set 2 using the proposed algorithm. The regions are overlaid on the raw data from Figure 5.58.

the screen, both before and after it divides less than halfway through the tracking run. As neither algorithm takes cell division into account, it is good to know that the proposed algorithm is able to retarget on the nucleus of one of the daughter cells. Once again, we see that after Cell Tracker runs close nuclei together, the resulting estimated cytoplasmic boundaries are poorly correlated with the boundaries of the actual cells.

The tracking results are visually better than Cell Tracker's. Although there are nonassigned forground pixels encroaching on actuall cytoplasmic pixels, this is most likely an issue with the blob detection algorithm as opposed to the base tracking algorithm. One problem we do see is that in the two nuclei at the top of the image, it seems that the yellowish nuclei on the right has a tendency to encroach on the brown nucleus immediately to its left, eventually pushing it completely out of position.

Experimental Data Set 3

The third experimental data set was also provided by Dr. Brasier's research group. It has a number of the same trials as the second data set, although there are no nuclei near each other. One problem which can clearly be seen is a distinct lack of contrast in the initial image, especially between the nuclear and cytoplasmic intensities. This makes the initial segmentation extremely difficult for the proposed algorithm. Selected frames from this data set are shown in Figure 5.61.

As can be seen from the first frame in Figure 5.62, the initial segmentation from

Figure 5.60 : Visualized tracking results from experimental data set 2 using Cell Tracker. The region boundaries are overlaid on the raw data from Figure 5.58.

Figure 5.61 : Raw data from the third experimental time series. Data provided by Dr. Allan Brasier.

the proposed algorithm is reasonable. Compared to Cell Tracker's segmentation in Figure 5.63, the proposed algorithm has a better initial segmentation. Although there several problems with the segmentation in the proposed algorithm, including the lack of a nucleus at the center of the screen and the incorrectly sized pink nucleus in the top corner, the segmentation from Cell Tracker fails to detect several cells, leading to overlarge cytoplasmic boundaries and permanent "nuclear drift."

The tracking results for the proposed algorithm, however, were not as stellar as other runs. This time series had an extended translocation sequence, in which the nuclei had a similar distribution to the cytoplasm for nearly 30 frames. During this time, the proposed algorithm managed to lose the nucleus for both the green cell (dark blue nucleus) and the orange cell (green nucleus), errors from which the proposed algorithm was never able to recover. In addition, we also see the unassigned foreground region encroaching into several cells in the image, especially the dark cell in the upper right corner of the image frame. This is likely due to enough of the cytoplasmic intensities falling below the L_2E threshold to disconnect them from the cytoplasmic blob. This issue would result in the nucleus no longer being associated with that foreground cellular area. However, we do notice that the proposed algorithm was able to successfully track the pink cell in the lower left corner of the image, as well as the pink nucleus for the blue cell in the center of the image.

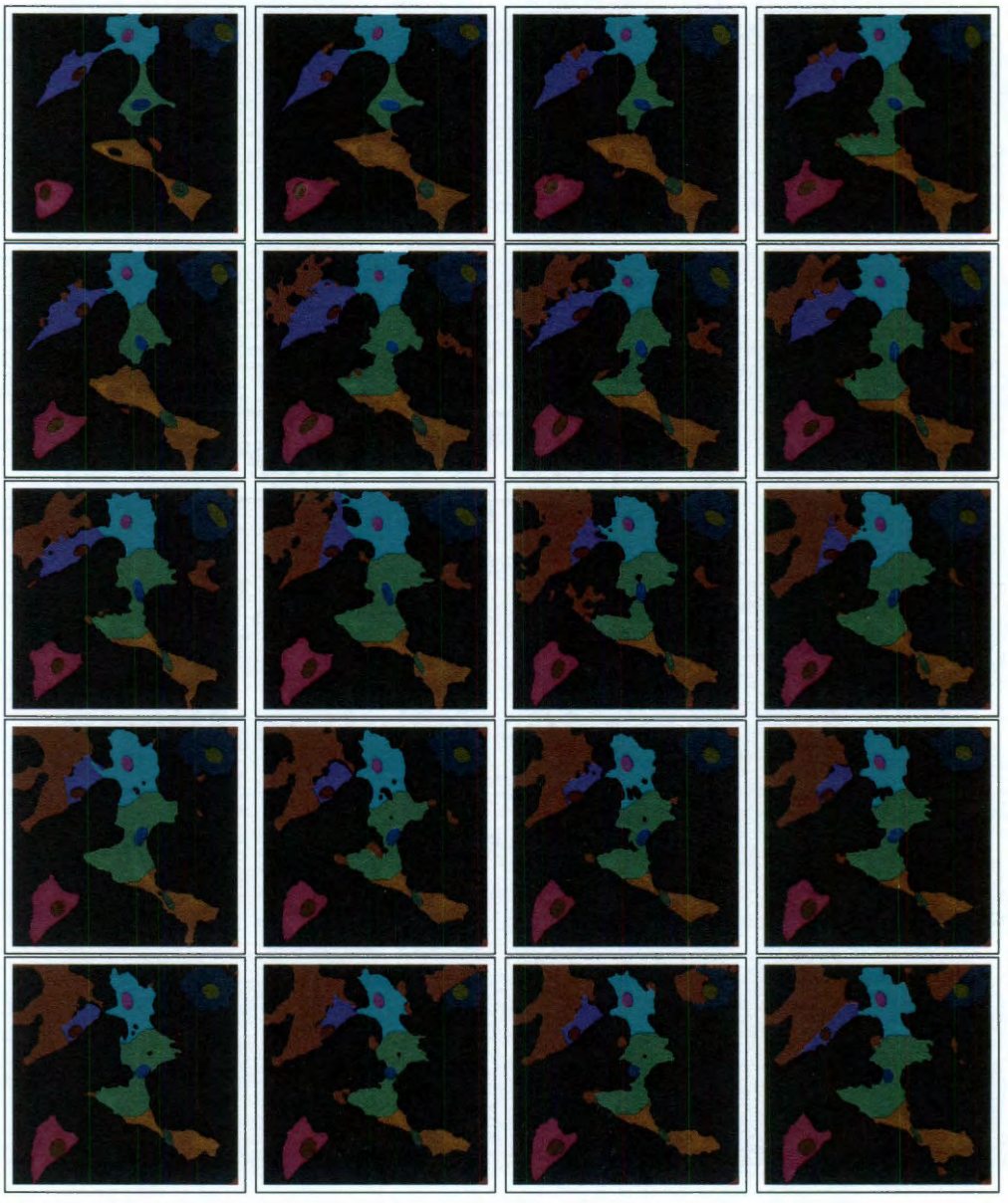


Figure 5.62 : Visualized tracking results from experimental data set 3 using the proposed algorithm. The regions are overlaid on the raw data from Figure 5.61.

Figure 5.63 : Visualized tracking results from experimental data set 3 using Cell Tracker. The region boundaries are overlaid on the raw data from Figure 5.61.

5.2.3 Comparison with Manual Tracking Results

In addition to quantified error against our simulation scheme and visual comparisons with Cell Tracker using experimental data, we also compared the results of the proposed algorithm against data that had been tracked entirely by hand using Cell Tracker's interactive interface. As the goal of the experiments we were to calculate the translocation of protein from the cytoplasm to the nucleus, we chose to use the integrated region intensity as our measure of comparison. It must be noted, however, that this particular measure of comparison is highly sensitive to the number of pixels assumed to be in each region. Therefore, the comparisons will consist of subjective visual classification based mostly on the shape of the region integrated intensity plots.

We claim that a cell fails to track when the cytoplasm becomes dissociated from the nucleus of the same cell. This is recognizable by frames with integrated cytoplasmic intensity of zero, as depicted in Figure 5.64. A cell is considered to be tracked poorly when neither the shape of the integrated cytoplasmic intensities nor the shape of the integrated nuclear intensities matches the manually tracked results for the same cell. A good example of a poor tracking result can be seen in Figure 5.65. We consider a medium tracking result to be an instance in which either the shape of the integrated cytoplasmic intensities or the shape of the integrated nuclear intensities matches the shape of the manually tracked results, as depicted in Figure 5.66. A good tracking result is when the shape is matched on both but the total intensity level does not between frames or one of the two intensity plots produced by the proposed algorithm

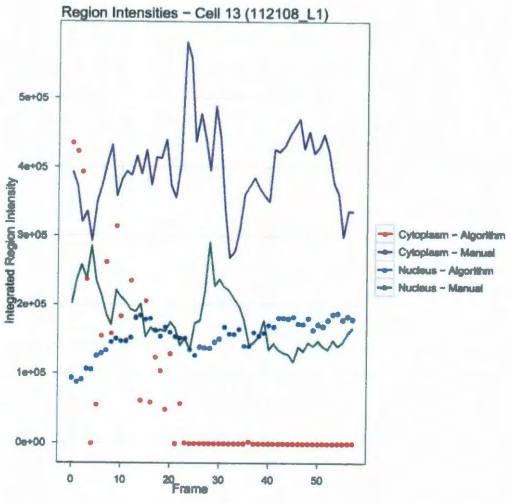


Figure 5.64 : An example of a failed tracking run. Solid lines are manual tracking results. Dots are results from the proposed algorithm.

is effectively equal to the appropriate intensity plot from the manual tracking results. An example is given in Figure 5.67. A very good result would be an instance where the shapes match for both regions and the total intensity levels match for at least one of the regions, of which an example is given in Figure 5.68.

In the six manually tracked time series that the proposed algorithm was able to open, there were a total of 39 manually tracked cells, the analysis of which required nearly three months. The proposed algorithm detected and tracked 91 cells in the same six time series to some degree in a total of approximately five hours. Of these 91 tracked cells, 29 matched cells that were tracked manually. Of these 29 matched cells, we found that 2 cells that failed to track, 2 cells tracked poorly, 4 cells had

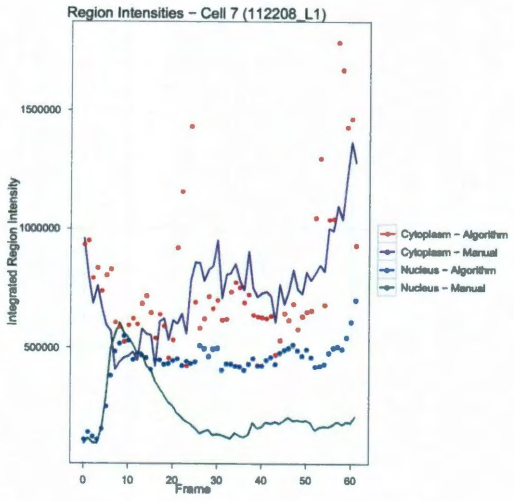


Figure 5.65 : An example of a poor tracking run. Solid lines are manual tracking results. Dots are results from the proposed algorithm.

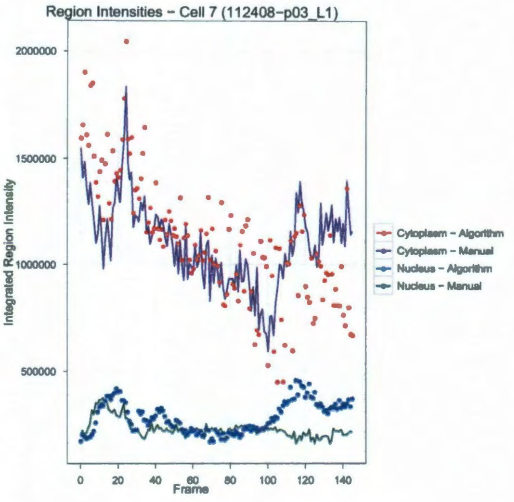


Figure 5.66 : An example of a medium tracking run. Solid lines are manual tracking results. Dots are results from the proposed algorithm.

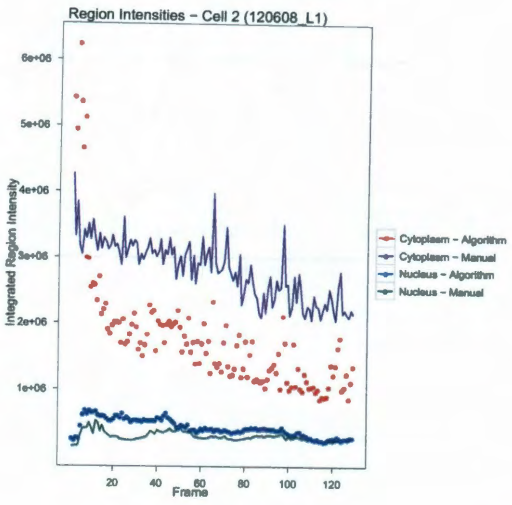


Figure 5.67 : An example of a good tracking run. Solid lines are manual tracking results. Dots are results from the proposed algorithm.

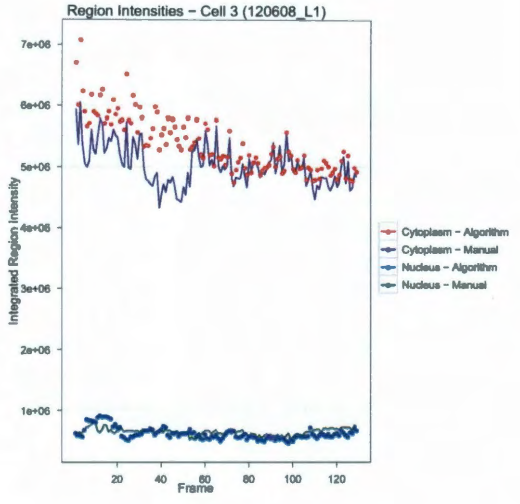


Figure 5.68 : An example of a very good tracking run. Solid lines are manual tracking results. Dots are results from the proposed algorithm.

medium tracking results, 11 cells had good tracking results, and 10 cells had very good tracking results.

Chapter 6

Conclusions

We have shown here a proposed improvement to Cell Tracker's segmentation and tracking algorithm. The proposed segmentation algorithm uses a level set based approach to iteratively determine foreground and background using the distribution of pixel intensities. A rapid ellipse detection and scoring algorithm is used to determine the location of nuclei within the image. Once these locations are known, the foreground pixels are divided into the the cytoplasm of each cell.

The proposed tracking algorithm improves on Cell Tracker's version of particle filter by introducing not only a second source of edges for the image, but also by obtaining a reasonable estimate for the intrinsic position change. This is done by implementing a novel two-step algorithm for calculating the optical flow between subsequent images in a region-dependent manner. This novel method upholds the base assumptions of the optical flow calculation methods while allowing for the relaxation of the brightness constraint.

The proposed tracking algorithm handles cells that were not in the field of view at the beginning of the experiment by labelling them as unclassified foreground items. This avoids one of the largest problems in Cell Tracker - the tendency for cytoplasm boundaries to expand and cover adjacent cells. By keeping a rough idea of where

these unclassified foreground objects are located, prevention of cytoplasmic boundary expansion can be ensured.

We have shown that in both simulation-based studies and in real data that the proposed algorithm fails to suffer many of the drawbacks seen in results using the Cell Tracker software. However, the proposed algorithm has a large number of parameters that need to be tuned for a truly successful tracking. Because the proposed algorithm is highly stochastic, the results are likely to vary between runs, as opposed to the Cell Tracker software, in which extremely similar if not exact results will be obtained each time the software analyzes the same data series.

In addition, it is obvious from many of the resulting images that the estimates of the nuclear ellipse parameters are not always correct. There are several causes for the error expressed in these situations. First, the scores generated for the particle filter are based on either the detected edges in the raw image data or the boundaries in the estimated region assignments. Due not only to the noise in the cytoplasm of the cell, but also the constantly dark nucleoli, it is possible that the particle filter algorithm is favoring these "noise" edges as opposed to the true nuclear edge. If this is the case, then the best way to solve this problem would be to modify the tracking parameters, especially the number of edge normals used to score the potential ellipses in the particle filter algorithm.

Other current drawbacks to the proposed algorithm include the possibility of numerical instability in the optical flow calculation. Because both ordinary and total

least squares calculations are used at various points in the tracking process, the lack of invertible matrices can cause critical failures in the proposed algorithm. It may be possible to overcome this problem by using singular value decomposition when solving the least squares problems, but this fix is not entirely successful.

Another drawback is the lack of defunct nuclei removal, that is a failure to remove the tracking nuclear area when the true nucleus leaves the image window. One possible way to solve this would be to see whether the forecasted center of the ellipse is outside the image window. If that is the case, then that nucleus could simply be removed from the tracking queue. However, such a method would take a significant modification to the current code base, which is why it has not yet been implemented.

The final drawback to the proposed algorithm could also be considered a feature. The proposed tracking algorithm is able to detect nuclei which lay partially outside an image. However, these nuclei are rarely tracked correctly due to the limiting change in the ellipse aspect ratio. By using a linear least squares approach to fitting the ellipses, less than optimal fits are often calculated. In cases where only portions of the ellipse can be seen, the least squares fit will match the visual edge of the ellipse as much as possible while still staying within the image window. One method of resolving this issue is to use a geometric fit (minimize squared distance) to determine the ellipse coefficients. However, this method requires solving nonlinear least squares problems, which have been bypassed in the interests of application speed.

At present, we have found the best way to handle a tracking run is to use a fairly

large smoothing and local threshold radius parameter for the level set segmentation procedure, as this reduces the the number of extraneous edge pixels within the cells themselves. The nuclear detection parameters should be fairly unrestrictive, which will allow for weakly defined nuclei to be detected. However, this will also result in a large number of false nuclei. We have found that it is worth waiting until the first frame is being tracked, then cancelling the program and examining the output for the first frame. At that point, the user can manually remove lines from the intermediate file that defines the ellipses, leaving only the true ellipses to be tracked.

We have several possibilities of directions for future research in this area besides what we have described above. The first possibility is to implement an on-the-fly correction scheme. This type of scheme would make the proposed tracking algorithm a mixture of forward- and backward-looking algorithms. A correction scheme would be comprised of two portions. The first portion would be a method to determine when the nuclear and cytoplasmic intensities would be sufficiently separated to allow for resegmentation of the image. We would correspond the newly found nuclear and cytoplasm locations with the current predictions. Once these correspondences are known, the current predictions could be updated with the newly-segmented values.

A second possibility of future research would be the development of a graphical user interface (GUI) for both the generation of parameter files and frame-by-frame result tuning. This GUI would allow for the user to input via mouse the known locations of nuclei to be tracked, as well as allow for easy parameter tuning. The

user would then also be able to easily remove any extraneous nuclei, preventing the tainting of the results. The frame-by-frame result tuning would allow the program to predict the subsequent frame, which the user could then more easily correct by mouse interaction, with the corrections carried forward into later frames.

Other possibilities of future research include allowances for cell division within the tracking algorithm, and improving our foreground blob detection algorithm. The improvement of the foreground blob algorithm will ensure that there are not odd diagonal stripes of "other foreground region" in the tracking results of different cytoplasm areas, as have been seen in several simulations. The allowance for cell division within the tracking algorithm is currently a contentious issue, with some researchers insisting that the results from daughter cells would be unusable, and other researchers claiming that as much data as can be generated will be helpful. In either case, a method of handling this allowance is currently unknown.

Bibliography

[1] Vikas Agrawal, Chu Zhang, Allan D Shapiro, and Prasad S Dhurjati, *A dynamic mathematical model to clarify signaling circuitry underlying programmed cell death control in arabidopsis disease resistance.*, Biotechnol Prog **20** (2004), no. 2, 426–442 (eng).

[2] Natalie Arkus, *A mathematical model of cellular apoptosis and senescence through the dynamics of telomere loss.*, J Theor Biol **235** (2005), no. 1, 13–32 (eng).

[3] Elife Z Bagci, Yoram Vodovotz, Timothy R Billiar, Bard Ermentrout, and Ivet Bahar, *Computational insights on the competing effects of nitric oxide in regulating apoptosis.*, PLoS One **3** (2008), no. 5, e2249 (eng).

[4] David Baraff and Andrew Witkin, *Large steps in cloth simulation*, SIGGRAPH, 1998.

[5] Jasper Bedaux, *C++ mersenne twister pseudo-random number generator*, 2003.

[6] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt, *Openmesh - a generic and efficient polygon mesh data structure*, 2002.

[7] Dean Bottino, Alexander Mogilner, Tom Roberts, Murray Stewart, and Geo, *How nematode sperm crawl*, Journal of Cell Science **115** (2002), 367–384.

[8] Allan R Brasier, *The nf-κb regulatory network.*, Cardiovasc Toxicol **6** (2006), no. 2, 111–130 (eng).

[9] Richard L. Burden and J. Douglas Faires, *Numerical analysis*, Brooks/Cole, 2010.

[10] Laurence Calzone, Laurent Tournier, Simon Fourquet, Denis Thieffry, Boris Zhivotovsky, Emmanuel Barillot, and Andrei Zinovyev, *Mathematical modelling of cell-fate decision in response to death receptor engagement.*, PLoS Comput Biol **6** (2010), no. 3, e1000702 (eng).

[11] John Canny, *A computational approach to edge detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence **8** (1986), 679–714.

[12] David E. Cardoze, Alexandre Cunha, Gary L. Miller, Todd Phillips, and Noel Walkington, *A bézier-based approach to unstructured moving meshes*, Symposium on Computational Geometry, 2004, pp. 310–319.

[13] Paolo Cazzaniga, Dario Pescini, Daniela Besozzi, Giancarlo Mauri, Sonia Colombo, and Enzo Martegani, *Modeling and stochastic simulation of the ras/camp/pka pathway in the yeast saccharomyces cerevisiae evidences a key regulatory function for intracellular guanine nucleotides pools.*, J Biotechnol **133** (2008), no. 3, 377–385 (eng).

[14] Dmitry Chetverikov, *A simple and efficient algorithm for detection of high curvature points in planar curves*, Computer Analysis of Images and Patterns, 2003, pp. 746–753.

[15] Alex Yong Sang Chia, Maylor K.H. Leung, How-Lung Eng, and Susanto Rahardja, *Ellipse detection with hough transform in one dimensional parametric space*, IEEE International Conference on Image Processing, vol. 5, 2007, pp. 333–336.

[16] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer, *Real-time tracking of non-rigid objects using mean shift*, Computer Vision and Pattern Recognition, IEEE Computer Society Conference on, vol. 2, 2000.

[17] P. Delanges, J. Benois, and D. Barba, *Active contours approach to object tracking in image sequences with complex background*, Pattern Recognition Letters **16** (1995), no. 2, 171–178.

[18] Boris Delaunay, *Sur la sphere vide*, Bulletin of Academy of Sciences of the USSR (1934), 793–800.

[19] Arnaud Doucet, De Nando Freitas, and Neil Gordon (eds.), *Sequential monte carlo methods in practice.*, Springer, 2001.

[20] Ardith W El-Kareh, Rachel E Labes, and Timothy W Secomb, *Cell cycle checkpoint models for cellular pharmacology of paclitaxel and platinum drugs.*, AAPS J **10** (2008), no. 1, 15–34 (eng).

[21] David J. Fleet and Yair Weiss, *Mathematical models in computer vision: The handbook*, ch. Optical Flow Estimation, pp. 239–258, Springer, 2005.

[22] Avner Friedman, Bei Hu, and Chiu-Yen Kao, *Cell cycle control at the first restriction point and its effect on tissue growth.*, J Math Biol **60** (2010), no. 6, 881–907 (eng).

[23] Michael Garland, *Quadric-based polygonal surface simplification*, Ph.D. thesis, Carnegie Mellon University, 1999.

[24] Rafael C. Gonzales and Richard E. Woods, *Digital image processing*, third ed., Prentice-Hall, 2008.

[25] N.J. Gordon, D.J. Salmond, and A.F.M. Smith, *Novel approach to nonlinear/non-gaussian bayesian state estimation*, Radar and Signal Processing, IEE Proceedings F, vol. 140, 1993, pp. 107–113.

[26] Burkhard Haefner, *Nf-kappa b: arresting a major culprit in cancer.*, Drug Discov Today **7** (2002), no. 12, 653–663 (eng).

[27] Radim Halir and Jan Flusser, *Numerically stable direct least squares fitting of ellipses*, International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media **1** (1998), 125–132.

[28] H. W. Haussecker and D. J. Fleet, *Computing optical flow with physical models*

of brightness variation, IEEE Computer Society Conference on Computer Vision and Pattern Recognition **2** (2000), 2760.

[29] Berthold K. P. Horn and Brian G. Schunck, *Determining optical flow*, Tech. report, Massachusetts Institute of Technology Artifical Intelligence Laboratory, 1980.

[30] Michael Isard and Andrew Blake, *Condensation - conditional density propagation for visual tracking*, International Journal of Computer Vision **29** (1998), no. 1, 5–28.

[31] Benot Jacob and Gal Guennebaud, *Eigen*, Software, 2009.

[32] Khuloud Jaqaman, Dinah Loerke, Marcel Mettlen, Hirotaka Kuwata, Sergio Grinstein, Sandra L Schmid, and Gaudenz Danuser, *Robust single-particle tracking in live-cell time-lapse sequences.*, Nat Methods **5** (2008), no. 8, 695–702 (eng).

[33] Thouis R. Jones, Anne Carpenter, and Polina Goll, *Voronoi-based segmentation of cells on image manifolds*, in Proc. ICCV Workshop on Computer Vision for Biomedical Image Applications, 2005, pp. 535–543.

[34] Thouis R Jones, In Han Kang, Douglas B Wheeler, Robert A Lindquist, Adam Papallo, David M Sabatini, Polina Golland, and Anne E Carpenter, *Cellprofiler analyst: data exploration and analysis software for complex image-based screens.*, BMC Bioinformatics **9** (2008), 482 (eng).

[35] Junmo Kim, John W. Fisher III, Anthony Yezzi, Mujdat Cetin, and Alan S. Willsky, *A nonparametric statistical method for image segmentation using information theory and curve evolution*, IEEE Transactions on Image Processing **14** (2005), no. 10, 1486–1502.

[36] Peter Kovesi, *Peter's functions for computer vision*, Website, 2009.

[37] H.W. Kuhn, *The hungarian method for the assignment problem*, Naval Research Logistics Quarterly **2** (1955), 83–97.

[38] C.L. Lam and S.Y. Yuen, *An unbiased active contour algorithm for object tracking*, Pattern Recognition Letters **19** (1998), no. 5-6, 491–498.

[39] Chunming Li, Chenyang Xu, Changfeng Gui, and Martin D. Fox, *Level set evolution without re-initialization: A new variational formulation*, Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, pp. 430–436.

[40] Fuhai Li, Xiaobo Zhou, and Stephen Wong, *Optimal live cell tracking for cell cycle study using time-lapse fluorescent microscopy images*, Machine Learning in Medical Imaging (Fei Wang, Pingkun Yan, Kenji Suzuki, and Dinggang Shen, eds.), Lecture Notes in Computer Science, vol. 6357, Springer, 2010, pp. 124–131.

[41] Fuhai Li, Xiaobo Zhou, Jinmin Zhu, Jinwen Ma, Xudong Huang, and Stephen TC Wong, *High content image analysis for human h4 neuroglioma cells exposed to cuo nanoparticles*, BMC Biotechnology **7** (2007), no. 1, 66.

[42] Tomasz Lipniacki and Marek Kimmel, *Deterministic and stochastic models of nf-κb pathway*, Cardiovascular Toxicology **7** (2007), 215–235.

[43] Alfred J. Lotka, *Analytical note on certain rhythmic relations [n organic systems*, Proceeds of the National Acadamy of Sciences **6** (1920), no. 7, 410–415.

[44] B. D. Lucas and T. Kanade, *An iterative image registration technique with an application to stero vision*, Proceedings of Imge Understanding Workshop, 1981, pp. 121–130.

[45] M. Matsumoto and T. Nishimura, *Mersenne twister: A 623-dimensionallly equidistributed uniform pseudo-random number generator*, ACM Transactions on Modeling and Computer Simulation **8** (1998), no. 1, 3–30.

[46] M Matyka and M Ollila, *"a pressure model for soft body simulation"*, Proceedings of Sigrad, UMEA (2003).

[47] A. Mogilner and D. W. Verzi, *A simple 1-d physical model for the crawling nematode sperm cell*, Journal of Statistical Physics **110** (2003), 1169–1189.

[48] Alexander Mogilner and George Oster, *The physics of lamellipodial protrusion*, Eur Biophys J **25** (1996), 47–53.

[49] Vlad I. Morariu, Balaji Vasan Srinivasan, Vikas C. Raykar, Ramani Duraiswami, and Larry S. Davis, *Automatic online tuning for fast gaussian summation*, Advances in Neural Information Processing Systems (NIPS), 2008.

[50] David M. Mount and Sunil Arya, *Ann: A library for approximate nearest neighbor searching*, Software, 2006.

[51] Matthias M uller-Fischer, *Real time physics*, ACM SIGGRAPH, vol. Class Notes, 2008.

[52] James D. Murray, *Mathematical biology: I. an introduction*, Springer, 2005.

[53] Thanh Minh Nguyen, Siddhant Ahuja, and Q.M. Jonathan Wu, *A real-time ellipse detection based on edge grouping*, IEEE International Conference on Systems, Man and Cybernetics, 2009.

[54] M.S. Nikulin (ed.), *Hellinger distance*, Springer, 2001.

[55] Nokia, *Qt: A cross-platform application and ui framework*, Software, 2009.

[56] M. N. Obeyesekere, S. O. Zimmerman, E. S. Tecarro, and G. Auchmuty, *A model of cell cycle behavior dominated by kinetics of a pathway stimulated by growth factors.*, Bull Math Biol **61** (1999), no. 5, 917–934 (eng).

[57] Stanley Osher and James A. Sethian, *Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulations*, Journal of Computational Physics **79** (1988), 12–49.

[58] Christian H. Schreiber, Murray Stewart, and Thomas Duke, *Simulation of cell motility that reproduces the forcevelocity relationship*, Proceedings of the National Academy of Sciences **107** (2010), no. 20, 9141–9146.

[59] David W. Scott, *Parametric statistical modeling by minimum integrated square error*, Technometrics **43** (2001), no. 3, 274–285.

[60] Feimo Shen, Louis Hodgson, Andrew Rabinovich, Olivier Pertz, Klaus Hahn, and Jeffrey H. Price, *Functional proteometrics for cell migration*, Cytometry **69** (2006), 563–572.

[61] Hailin Shen, Glyn Nelson, David E Nelson, Stephnie Kennedy, David G Spiller, Tony Griffiths, Norman Paton, Stephen G Oliver, Richael R.H. White, and Douglas B Kell, *Automated tracking of gene expression in individual cells and cell compartments*, Journal of the Royal Society Interface **3** (2006), 787–794.

[62] Jonathan Richard Shewchuk, *Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator*, Applied Computational Geometry: Towards Geometric Engineering (Ming C. Lin and Dinesh Manocha, eds.), Lecture Notes in Computer Science, vol. 1148, Springer-Verlag, May 1996, From the First ACM Workshop on Applied Computational Geometry, pp. 203–222.

[63] William R. Taylor, Zoe Katsimitsoulia, and Alexei Poliakov, *Simulation of cell movement and interaction*, Journal of Bioinformatics and Computational Biology **9** (2011), no. 1.

[64] Paolo Ubezio, Monica Lupi, Davide Branduardi, Paolo Cappella, Edoardo Cavallini, Valentina Colombo, Giada Matera, Claudia Natoli, Daniela Tomasoni, and

Maurizio D'Incalci, *Quantitative assessment of the complex dynamics of g1, s, and g2-m checkpoint activities.*, Cancer Res **69** (2009), no. 12, 5234–5240 (eng).

[65] Tamiki Umeda and Kei Inouye, *Possible role of contact following in the generation of coherent motion of dictyostelium cells*, Journal of Theoretical Biology **219** (2002), 301–308.

[66] A.W. van der Vaart, *Asymptotic statistics (cambridge series in statistical and probabilistic mathematics)*, Cambridge University Press, 1998.

[67] Dale Varberg, Edwin J. Purcell, and Steven E. Rigdon, *Calculus*, eigth edition ed., Prentice Hall, 2000.

[68] C. Vieren, F. Cabestaing, and J.G. Postaire, *Catching moving-objects with snakes for motion tracking*, Pattern Recognition Letters **16** (1995), no. 7, 679–685.

[69] Vito Volterra, *Fluctuations in the abundance of a species considered mathematically*, Nature **118** (1926), 558–60.

[70] Walaber, *Jellyphysics library*, Web, June 2008.

[71] John Weaver, *Munkres (hungarian) algorithm in c++*, Web, 2007.

[72] Jie Yao, Nawwaf Kharma, and Peter Grogono, *A multi-population genetic algorithm for robust and fast ellipse detection*, Pattern Analysis Applications **8** (2005), 149–162.

[73] Rong Zhou, *Numerical simulation of cell movement*, Ph.D. thesis, University of Connecticut, 2008.

[74] Zhike Zi, Kwang-Hyun Cho, Myong-Hee Sung, Xuefeng Xia, Jiashun Zheng, and Zhirong Sun, *In silico identification of the key components and steps in ifn-gamma induced jak-stat signaling pathway.*, FEBS Lett **579** (2005), no. 5, 1101–1108 (eng).

[75] Barbara Zitova and Jan Flusser, *Image registration methods: a survey*, Image and Vision Computing **21** (2003), 977–1000.

Appendix A

Derivations

A.1 Force Jacobian Calculation

A.1.1 Jacobian of Spring Forces

Examine $\boldsymbol{K}$ from Chapter 4.4.2. For the majority of the points in the system, there is only a single force acting upon it - the force based on the springs attached to that node. Recall from (4.3) that for any pair of connected nodes A and B, the force acting on the two nodes from the spring connecting them is given by

$$F^s_{A,B} = k\frac{\boldsymbol{x}_B - \boldsymbol{x}_A}{||\boldsymbol{x}_B - \boldsymbol{x}_A||}(||\boldsymbol{x}_B - \boldsymbol{x}_A|| - l_0), \quad F^s_{B,A} = -F^s_{A,B}.$$

The following calculates the Jacobian of $F^s_{A,B}$ with respect to $\boldsymbol{x}_A$.

$$\begin{aligned}\frac{\partial F^s_{A,B}}{\partial \boldsymbol{x}_A} &= \frac{\partial}{\partial \boldsymbol{x}_A} k\left((\boldsymbol{x}_B - \boldsymbol{x}_A) - l_0\frac{\boldsymbol{x}_B - \boldsymbol{x}_A}{||\boldsymbol{x}_B - \boldsymbol{x}_A||}\right) \\ &= k\left(-\boldsymbol{I} - \frac{l_0}{l}\left(\boldsymbol{I} - \frac{(\boldsymbol{x}_B - \boldsymbol{x}_A)(\boldsymbol{x}_B - \boldsymbol{x}_A)^T}{l^2}\right)\right). \end{aligned} \tag{A.1}$$

Notice that this portion of the Jacobian is a 2×2 matrix, which will be denoted $\boldsymbol{K}_{A,A}$, as the derivative of the force acting on node A with respect to the position of node A. It is easy to see that

$$\boldsymbol{K}_{A,A} = -\boldsymbol{K}_{A,B} = -\boldsymbol{K}_{B,A} = \boldsymbol{K}_{B,B}.$$

For simplicity, the Jacobian of the nuclear vertices are handled as standary cytoplasmic vertices. Also notice that for $A \in \phi$, the contribution to $\boldsymbol{K}_{A,A}$ is calculated as in (A.1).

A.1.2 Jacobian of Pressure Forces

However, the boundary points are also susceptible to pressure-based forces. Recall from (4.10) and (4.7) that the total area enclosed by the cell is given by

$$A(S) = \sum_{l=1}^{L} \frac{1}{2} |x_{B_l} - x_{A_l}| * \boldsymbol{n}_x(l) * |\boldsymbol{l}|,$$

and the magnitude of the pressure force on any given edge is given by

$$F^p_{A,B} = (||\boldsymbol{x}_B - \boldsymbol{x}_A|| k_P) \left(\frac{1}{\boldsymbol{A}} - \frac{1}{\boldsymbol{A}_0} \right).$$

Now, assuming that the border points are handled in clockwise order, so that point A comes before point B, the outer normal vector to the edge $\overline{AB}$ is given by

$$\begin{pmatrix} y_B - y_A \\ x_A - x_B \end{pmatrix},$$

and it is normalized by the length of $\overline{AB}$. Hence, the pressure force equation is given by

$$F^p_{A,B} = k_P \begin{pmatrix} y_B - y_A \\ x_A - x_B \end{pmatrix} \left(\frac{1}{\boldsymbol{A}} - \frac{1}{\boldsymbol{A}_0} \right). \tag{A.2}$$

To calculate the Jacobian for the pressure forces, each derivative must be examined separately. First calculate the derivative of $A(S)$ with respect to the change in a boundary point $\boldsymbol{x}_A$.

From (4.10), examine $\boldsymbol{n}_x(l)$. The boundary points are assumed to be in clockwise order as described above, so that $C < A < B$. For boundary point $\boldsymbol{x}_A$, there are exactly two boundary edges which $\boldsymbol{x}_A$ affects, $\overline{CA}$ and $\overline{AB}$. Thus $\partial A/\partial \boldsymbol{x}_A$ collapses into two terms,

$$\begin{aligned}\frac{\partial A(S)}{\partial \boldsymbol{x}_A} &= \frac{1}{2}|x_A - x_C|\frac{(y_C - y_A)}{||\boldsymbol{x}_A - \boldsymbol{x}_C||}||\boldsymbol{x}_A - \boldsymbol{x}_C|| \\ &\quad + \frac{1}{2}|x_B - x_A|\frac{(y_A - y_B)}{||\boldsymbol{x}_B - \boldsymbol{x}_A||}||\boldsymbol{x}_B - \boldsymbol{x}_A|| \\ &= \frac{1}{2}|x_A - x_C|(y_C - y_A) + \frac{1}{2}|x_B - x_A|(y_A - y_B)\end{aligned}$$

where $y_C - y_A$ is the x component of the normal for the first edge and $y_A - y_B$ is the x component of the normal for the second edge.

Now, for any given boundary node $\boldsymbol{x}_A$,

$$\frac{\partial A(S)}{\partial x_A} = -\frac{1}{2}\frac{|x_B - x_A|}{x_B - x_A}(y_A - y_B) + \frac{1}{2}\frac{|x_B - x_A|}{x_B - x_A}(y_C - y_A)$$

Similarly,

$$\frac{\partial A(S)}{\partial y_i} = \frac{1}{2}|x_B - x_A| - \frac{1}{2}|x_B - x_A|.$$

It can be shown that

$$\frac{\partial A(S)}{\partial \boldsymbol{x}_A} = \frac{1}{2}\begin{pmatrix} \frac{y_A - y_B}{x_A - x_B} & \frac{y_C - y_A}{x_A - x_C} \\ 1 & -1 \end{pmatrix}\begin{pmatrix} |x_B - x_A| \\ |x_A - x_C| \end{pmatrix} \tag{A.3}$$

Now force of pressure on A can be considered. Note the pressure force applied on A is a sum of the forces from the two edges leading from A – $\overline{AB}$ and $\overline{AC}$ – where A,

B, and C are boundary points such that $C < A < B$ in a clockwise direction. Thus the pressure force acting on A is given by

$$\begin{aligned}\boldsymbol{F}_A &= k_P \begin{pmatrix} y_A - y_B \\ x_B - x_A \end{pmatrix} \left(\frac{1}{\boldsymbol{A}} - \frac{1}{\boldsymbol{A}_0}\right) + k_P \begin{pmatrix} y_C - y_A \\ x_A - x_C \end{pmatrix} \left(\frac{1}{\boldsymbol{A}} - \frac{1}{\boldsymbol{A}_0}\right) \\ &= k_P \begin{pmatrix} y_C - y_B \\ x_B - x_C \end{pmatrix} \left(\frac{1}{\boldsymbol{A}} - \frac{1}{\boldsymbol{A}_0}\right).\end{aligned}$$

From here, it is easily shown that if X is any other boundary node besides B or C, then

$$\frac{\partial \boldsymbol{F}_A}{\partial X} = \boldsymbol{F}'_{A,X} = -\frac{k_P}{A^2} \begin{pmatrix} y_C - y_B \\ x_B - x_C \end{pmatrix} \frac{\partial A(S)}{\partial \boldsymbol{x}_X}.$$

Similarly, if X is C, then

$$\boldsymbol{F}'_{A,X} = k_p \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \left(\frac{1}{A} - \frac{1}{A_0}\right) - \frac{k_P}{A^2} \begin{pmatrix} y_C - y_B \\ x_B - x_C \end{pmatrix} \frac{\partial A(S)}{\partial \boldsymbol{x}_X},$$

and if X is B, then

$$\boldsymbol{F}'_{A,X} = -k_p \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \left(\frac{1}{A} - \frac{1}{A_0}\right) - \frac{k_P}{A^2} \begin{pmatrix} y_C - y_B \\ x_B - x_C \end{pmatrix} \frac{\partial A(S)}{\partial \boldsymbol{x}_X},$$

Appendix B

Implementation Notes

This appendix includes details on the usage of the tracking and simulation algorithms. Appendix B.1 is a black-box overview of running the tracking algorithm. Appendix B.2 describes the parameter file formats and default values for the simulation discussed in Chapter 4. Appendix B.3 desribes the parameter file formats and default values for the tracking algorithm described in Chapter 3.

B.1 Proposed Algorithm Usage

The proposed algorithm is configured using the CMake build system (http://www.cmake.org). The program can then be built using any modern C++ compiler. The program is called via the command

```
track <parameter file>
```

where the parameter file is the filename of the main parameter file discussed in Appendix B.3.

This program requires all of the parameter files described in Appendix B.3 to be present. In addition, the inputs to the program can consist of a directory of ordered images (most image formats are acceptable) or a Carl Zeiss LSM-formatted file.

The outputs for this program in its present form consist of data in two different folders. The `segmentation` folder contains the data for intermediate segmentation steps. The file `localthreshold.png` is the result of the local thresholding operation. The `binary.png` file is the binary mask resulting from the level set algorithm in Chapter 3.3.2. The `EdgeMap.png` file is the masked edge map obtained from the `localthreshold.png` and `binary.png` images. The `ellipses.txt` file is a tab-delimited text file with the parameters of each ellipse on a single line. These parameters are in the order: 1) Center x coordinate, 2) Center y coordinate 3) Rotation (radians) 4) Major axis half-length 5) Minor axis half-length. This file can be used to add or remove ellipses from the segmentation. The last file, `cellPoints.txt` consists of all the points in each region. These points are separated by the character `|`.

The primary output directory is `results`. This folder contains the tracking results overlaid on the original data as PNG image files. This folder also contains a file called `means.txt`, which is a tab-delimited file containing the region name, the centroid of each nuclear ellipse from the initial frame, and the integrated region intensity for each frame.

If the tracking program is compiled with the `ANALYSIS` flag defined, then a folder entitled `analysis` will also be created. The `analysis` folder contains extraneous data to be used by a separate evaluation program, which is worthless unless paired directly with the simulation algorithm described in this paper.

```
<img width (int)> <img height (int)>
<number of cells (int)>
<min t (float)> <max t (float)> <delta t (float)> <num images (int)>
<output path (string)>
<filename for cell parameters (string)>
```

Figure B.1 : Parameter file structure for base simulation parameters

If the tracking program is compiled with the `DEBUG` flag defined, then a folder entitled `EXTRA` will be created. The `extra` folder contains data dumps of the estimated region classifications from Chapters 3.4.4 and 3.4.5, the optical flow vectors and the respective confidence intervals as calculated in Chapter 3.4.3, and the two edge images used in Chapter 3.4.6.

WARNING: The `means.txt` file *will* be overwritten on each new run. If runs need to be run both forward and backward, it is highly recommended to rename the `results` folder before starting the second run.

B.2 Simulation Parameter Values

The simulation parameter set currently resides in two files. The initial file describes the overall simulation paramters, such as the image height and width, the number of cells, and the parameters for the integration scheme. This file also contains the filename for the second parameter file. The format is given in Figure B.1. B.1 is a listing of the default parameter values for the primary simulation parameter file.

Parameter	Value	Parameter	Value
img width	512	delta t	0.01
img height	512	num images	300
number of cells	5	output path	.
min t	0	cell parameter file	cells
max t	50		

Table B.1 : Default parameters for the main simulation parameter file. The format is given as in Figure B.1

```
<min radius (int)> <max radius (int)> <corners (int)>
<nucleus min percentage (float)> <nucleus max percentage (float)> <nuc attempts (int)>
<min spring constant (float)> <max spring constant (float)> <damping constant (float)> <pressure constant (float)>
<mass mu (float)> <mass sd (float)>
<low intensity mean low (int)> <low intensity mean high (int)> <low intensity sd (int)>
<high intensity mean low (int)> <high intensity mean high (int)> <high intensity sd (int)>
<initial linearization tolerance (float)> <max quadric error (float)> <max nodes (int)>
<probability of movement (float)> <probability of continued movement (float)>
<jitter time mean (float)> <jitter time sd (float)> <jitter min (float)> <jitter max (float)>
<extend time mean (float)> <extend time sd (float)>
<center time mean (float)> <center time sd (float)>
<rear mean (float)> <rear sd (float)>
<rest mean (float)> <rest sd (float)>
<num pseudopods (int)> <width mean (float)> <width sd (float)> <direction variance (float)>
<extension multiplier min (float)> <extension multiplier max (float)>
<spring shrink multiplier (float)>
<min period multiplier (float)> <max period multiplier (float)>
```

Figure B.2 : Parameter file structure for cellular parameter file

Table B.2 : Default parameters for the cell parameter file. The format is given as in Figure B.2

Parameter	Value	Parameter	Value
Min radius	50	Probability continued movement	0.25
Max radius	90	Jitter time μ	1
Corners	6	Jitter time σ	0.1
Nucleus Min Percentage	0.15	Jitter min	0.5
Nucleus Max Percentage	0.25	Jitter max	2
Attempts to generate Nucleus	10	Extend time μ	2
Min spring constant	9	Extend time *sd*	0.2
Max spring constant	11	Center time μ	2
Damping constant	0	Center time σ	0.2
Pressure constant	10	Rear μ	1
Mass μ	50	Rear σ	0.1
Mass σ	5	Rest μ	1
Low intensity μ low	2	Rest σ	0.1
Low intensity μ high	4	Num Pseudopods	1
Low intensity σ	5	Width μ	30
High intensity μ low	80	Width σ	5
High intensity μ high	100	Direction variance	45

Continued on Next Page...

Table B.2 : Default parameters for the cell parameter file. The format is given as in Figure B.2

Parameter	Value	Parameter	Value
High intensity σ	40	Extension multiplier min	1.01
Initial linearization tolerance	2	Extension multiplier max	2.99
Max quadric error	500	Spring shrink multiplier	0.5
Maximum nodes	100	Min period multiplier	0.4
Probability movement	0.5	Max period multiplier	0.6

The second parameter file describes how the cells are generated and the parameters used for moving the cells. These parameters include distribution parameters for the length of time in each movement stage, the parameters for the high and low intensity distributions, and the cell creation parameters. The format of this file is listed in Figure B.2. The default parameters are listed in Table B.2.

B.3 Tracking Parameter Values

There are a number of parameter files used for implementing the tracking program, due to the large number of algorithms. These algorithms include the initial segmentation of the image, the calculation of optical flow and the particle filter algorithm itself. The base parameter file is used to read the data and specifies the current

```
<data location (file or directory name)> <progress indicator (string)>
<initial image (int) (0-indexed)> <channel (int) (0-indexed)> <direction (string)>
<level set parameter file (filename)>
<ellipse detection parameter file (filename)>
<cell tracking parameter file (filename)>
<optical flow parameter file (filename)>
```

Figure B.3 : Parameter file structure for primary tracking parameter file

Parameter	Value	Parameter	Value
Data location	data	Levelset parameter file	levelset.example
Progress indicator	none	Ellipse detection parameter file	nucdetect.example
Initial image	0	Cell tracking parameter file	tracking.example
Channel	0	Optical flow parameter file	of.example
Direction	forward		

Table B.3 : Default parameters for the main tracking parameter file. The format is given as in Figure B.3

segmentation stage as well as the locations of the other parameter files. The base parameter file format is listed in Figure B.3, with the default values shown in Table B.3.

The second parameter file for the proposed tracking program contains the parameters for the level set algorithm described in Chapter 3.3.2. These parameters include the weighting for the internal and external energy, as well as the weight placed on the

```
<data location (file or directory name)> <progress indicator (string)>
<initial image (int) (0-indexed)> <channel (int) (0-indexed)> <direction (string)>
<levelset parameter file (filename)>
<ellipse detection parameter file (filename)>
<cell tracking parameter file (filename)>
<optical flow parameter file (filename)>
```

Figure B.4 : Parameter file structure for level set algorithm

```
<min edge length (int)> <linearization tolerance (float)>
<angle threshold (degrees) (float)> <distance threshold (float)>
<local distance threshold (float)> <global distance threshold (float)>
<min major rad (float)> <max major rad (float)>
<min minor rad (float)> <max minor rad (float)>
<dark nuclei (true/false)>
<min candidate score (float)> <min candidate percentage (float)>
```

Figure B.5 : Parameter file structure for nuclear detection algorithm

curvature term. The file format is listed in Figure B.4 with the default values shown in Table B.4.

The third parameter file customizes the nuclear detection algorithm. The modifications to this file are primarily used to filter out unwanted ellipses. The file format is described in Figure B.5, with the default values given in Figure B.5.

The fourth parameter file customizes the different portions of the tracking algorithm. This file is used to handle the pixel assignment as described in Chapters 3.4.4-3.4.5 as well as the parameters for the edge detection and particle filter algorithms used in Chapter 3.4.6. This parameter file also includes values to handle the nonlinear optimization algorithm used to calculate the L_2E threshold in Chapter

Parameter	Value	Parameter	Value
Median blur radius	2	Curvature Weight	0.05
Local threshold radius	4	Kernel radius	7500
Border radius	5	Narrow band radius	5
Quantile radius	2	Iteration time change	10
Quantile target	0.75	Small change threshold	0.001
Iterations	50	Max small change	5
LR Weight	0.005	Num random points	5000

Table B.4 : Default parameters for the level set parameter file. The format is given as in Figure B.4

Parameter	Value	Parameter	Value
Min edge length	3	Maximum major radius	50
Linearization Tolerance	2.5	Minimum minor radius	10
Angle threshold (degrees)	40	Maximum minor radius	50
Distance threshold	4	Dark nuclei	true
Local distance threshold	10	Min candidate score	1e-10
Global distance threshold	30	Min candidate percentage	0.6
Minimum major radius	10		

Table B.5 : Default parameters for the nuclear detection parameter file. The format is given as in Figure B.5

```
<intensity diff std. dev (float)> <location diff mean (float)>
<OF fail spread (int)> <No assignment radius>
<NNBlur radius (int)> <NNBlur num Neighbors (int)>
<Canny std dev (float)> <Canny low thresh (int)> <Canny high thresh (int)>
<Num Random Ellipses (int)> <Rand distance (double)> <Rand Area change (double)> <Rand rotation change degrees (double)>
<Number of points for ellipse scoring (int)> <Max distance to test for edge (int)>
<L2E tolerance (double)> <L2E max iterations (int)>
```

Figure B.6 : Parameter file structure for tracking algorithm

```
<optical flow algorithm (String)>
<temporal smoothing method (String)> <temp smoothing rad (int)>
<spatial smoothing method (String)> <spatial smoothing rad (int)>
<parameter (String)> <value (?)>
```

Figure B.7 : Parameter file structure for optical flow algorithm

3.4.7. The format of the parameter file is shown in Figure B.6, with the default values listed in Table B.6.

The final parameter file provides the necessary parameters to calculate the optical flow for each frame. This file provides the name of the algorithm, details on the smoothing methods, and any other values necessary for the algorithm to run. The format of this file is given in Figure B.7, with the default parameter values for the proposed optical flow algorithm (Chapter 3.4.3) shown in Table B.7. Table B.8 shows the parameters required for each of the implemented optical flow algorithms.

Parameter	Value	Parameter	Value
Intensity diff std. dev.	5	Num random ellipses	500
Location diff mean	4	Rand distance	8
OF fail spread	10	Rand area change	0.1
No assignment radius	10	Rand rotation (degrees)	15
NNBlur radius	3	Number of points for scoring	15
NNBlur num Neighbors	50	Max distance	8
Canny std. dev	2	L_2E tolerance	0.01
Canny low thresh	35	L_2E max iterations	100
Canny high thresh	127		

Table B.6 : Default parameters for the tracking parameter file. The format is given as in Figure B.6

Parameter	Value	Parameter	Value
Temporal smoothing method	gaussian	Temporal smoothing radius	3
Spatial smoothing method	median	Spatial smoothing radius	5

Table B.7 : Necessary parameters for the optical flow parameter file. The format is given as in Figure B.7

Algorithm Code Name	Parameter	Value Type	Value
mine	smooth-alpha	Float	0.75
mine	of-rad	Int	2
mine	mean-rad	Int	1
mine	mean-points	Int	1000
horn-shuck	conv_value	Float	0.1
lucas-kanade	radius	Int	1
physical	degree	Int	2
physical	radius	Int	2

Table B.8 : Optional parameters for optical flow calculation, broken down by algorithm code. "mine" indicates the proposed algorithm, "horn-shuck" gives parameters for the Horn-Shunck algorithm, "lucas-kanade" represents the Lucas-Kanade least squares algorithm, and "physical" uses Haussecker and Fleet's algorithm.